Maryam Kamali

# Reusable Formal Architectures for Networked Systems

Turku Centre *for* Computer Science

TUCS Dissertations
No 162, September 2013

# Reusable Formal Architectures for Networked Systems

Maryam Kamali

*To be presented, with the permission of the Department of Information Technologies at Åbo Akademi University, for public criticism in Auditorium Gamma on September 13, 2013, at 12 noon.*

Turku Centre for Computer Science
Åbo Akademi University
Department of Information Technologies
Joukahainengatan 3-5, 20520 Åbo, Finland

2013

## Supervisors

Docent Luigia Petre
Department of Information Technologies
Åbo Akademi University
Joukahaisenkatu 3-5, 20520, Turku
Finland

Professor Kaisa Sere
Department of Information Technologies
Åbo Akademi University
Joukahaisenkatu 3-5 A, 20520, Turku
Finland

## Reviewers

Professor Tiziana Margaria-Steffen
Institute for Informatics
University of Potsdam
August Bebel Straße 89, 14482, Potsdam
Germany

Doctor Helen Treharne
Department of Computing
University of Surrey
Guildford, Surrey, GU2 7XH
UK

## Opponent

Professor Tiziana Margaria-Steffen
Institute for Informatics
University of Potsdam
August Bebel Straße 89, 14482, Potsdam
Germany

ISBN 978-952-12-2932-9
ISSN 1239-1883

*To my mom and dad*

تقدیم به پدر و مادر عزیزم

# Abstract

Today's networked systems are becoming increasingly complex and diverse. The current simulation and runtime verification techniques do not provide support for developing such systems efficiently; moreover, the reliability of the simulated/verified systems is not thoroughly ensured. To address these challenges, the use of formal techniques to reason about network system development is growing, while at the same time, the mathematical background necessary for using formal techniques is a barrier for network designers to efficiently employ them. Thus, these techniques are not vastly used for developing networked systems.

The objective of this thesis is to propose formal approaches for the development of reliable networked systems, by taking efficiency into account. With respect to reliability, we propose the architectural development of correct-by-construction networked system models. With respect to efficiency, we propose reusable network architectures as well as network development. At the core of our development methodology, we employ the abstraction and refinement techniques for the development and analysis of networked systems. We evaluate our proposal by employing the proposed architectures to a pervasive class of dynamic networks, i.e., wireless sensor network architectures as well as to a pervasive class of static networks, i.e., network-on-chip architectures. The ultimate goal of our research is to put forward the idea of building libraries of pre-proved rules for the efficient modelling, development, and analysis of networked systems. We take into account both qualitative and quantitative analysis of networks via varied formal tool support, using a theorem prover – the Rodin platform – and a statistical model checker – the SMC-Uppaal.

# Sammanfattning

Nätverkssystem blir numera allt mer invecklade och olika varandra. De existerande teknikerna för simulering och körtidsverifiering ger inget stöd för effektiv utveckling av nätverkssystem, och utöver detta kan simulerade/ verifierade systems pålitlighet inte fullständigt garanteras. För att möta dessa utmaningar har användningen av formella metoder för att resonera kring utveckling av nätverkssystem ökat, men den matematiska bakgrund som krävs för att använda formella metoder hindrar samtidigt nätverksutvecklare från att effektivt dra nytta av dessa metoder. För utveckling av nätverkssystem används dessa metoder därför inte i någon större utsträckning.

Målsättningen med denna avhandling är att föreslå formella tillvägagångssätt för utveckling av pålitliga nätverkssystem där effektiviteten tas i beaktande. Gällande pålitligheten föreslår vi modeller av nätverkssystem vars arkitektur utvecklas korrekt genom konstruktionen. När det gäller effektivitet föreslår vi nätverksarkitekturer och nätverksutveckling som kan återanvändas. Kärnan i vår utvecklingsmetodologi är att använda abstraktion- och preciseringstekniker för utveckling och analys av nätverkssystem. Vi utvärderar vårt förslag genom att använda de föreslagna arkitekturerna för en genomgripande klass av dynamiska nätverk, det vill säga trådlösa sensornätverk, och likaså en genomgripande klass av statiska nätverk, det vill säga arkitekturer för nätverk-på-chip. Det slutgiltiga målet med vår forskning är att föra fram idén om att bygga bibliotek innehållande på förhand bevisade regler för effektiv modellering, utveckling och analys av nätverkssystem. Vi tar i beaktande såväl kvalitativ som kvantitativ analys av nätverk via olika formella verktyg såsom Rodin-plattformen, för bevis av teorem, och SMC-Uppaal, för statistisk modellkontroll.

# Acknowledgements

First of all, I would like to express profound thanks to my supervisors Docent Luigia Petre and Prof. Kaisa Sere for their excellent advice and support, for their continuous encouragement, for the countless scientific discussions we have had, and for always being there to help, no matter the time or date. Furthermore, I wish to thank Prof. Tiziana Margaria-Steffen and Dr. Helen Treharne for their valuable reviews of this dissertation and for providing constructive comments that improved its quality. Particular thanks are due to Prof. Tiziana Margaria-Steffen for also agreeing to act as an opponent at the public defence of the thesis.

I am grateful to all the members of the Department of Information Technologies at Åbo Akademi University, especially my colleagues at the Distributed Systems Laboratory and Turku Centre for Computer Science for providing such a friendly atmosphere to work. In particular, I wish to thank Docent Linas Laibinis for his advice and encouragement as well as Dr. Anton Tarasyuk, Dr. Mats Neovius and Petter Sandvik for their help with practical matters and the Swedish version of the abstract. I would like to acknowledge Prof. Ansgar Fehnker's kind help and supervision during my stay at University of the South Pacific, Fiji. I extend my sincere thanks to Dr. Peter Höfner, Prof. Rob J. van Glabeek and Prof. Frank Cessar for their supervision and friendship during my stay at NICTA, Australia. I am very grateful to all my co-authors for the knowledge they shared with me.

I gratefully thank the Turku Centre for Computer Science and the Department of Information Technologies at Åbo Akademi University for the generous funding of my doctoral studies and travels. I would like to also acknowledge the Nokia foundation and the Ulla Tuomienen Foundation for granting me research scholarships that supported my work.

Finally, I would like to express my thankfulness to my family and my friends for their cheeriness and support throughout these years. I am especially grateful to my brothers, Ehsan and Morteza, and my dear sister, Mojgan, who continuously motivated me to keep the pace in my research. I would like to express my deepest gratitude to my dearest parents, Nahid and Ali, for their love and support during different phases of my life. This dissertation is dedicated to you.

Maryam Kamali
Åbo, August 2013

# List of original publications

I Maryam Kamali, Linas Laibinis, Luigia Petre, and Kaisa Sere. Formal Development of Wireless Sensor-Actor Networks, In *Science of Computer Programming (SCP) Journal*. Elsevier, 2012. DOI: 10.1016/j.scico.2012.03.002.

II Maryam Kamali, Linas Laibinis, Luigia Petre, and Kaisa Sere. A Distributed Implementation of a Network Recovery Algorithm, In *International Journal of Critical Computer-Based Systems (IJCCBS)*, Vol. 4, No. 1, pp. 45-68. Inderscience Publishers, 2013.

III Ansgar Fehnker, Peter Höfner, Maryam Kamali, and Vinay Mehta. Topology-based Mobility Model for Wireless Networks, In K. Joshi et al. (Eds.) *Proceedings of the 10th International Conference on Quantitative Evaluation of Systems conference - QEST 13*, Lecture Notes in Computer Science Vol. 8054, pp. 389-404, Springer-Verlag, 2013.

IV Peter Höfner, and Maryam Kamali. Quantitative Analysis of AODV and its Variants on Dynamic Topologies using Statistical Model Checking, In V. Braberman and L. Fribourg (Eds.) *Proceedings of the 11th International Conference on Formal Modeling and Analysis of Timed Systems - FORMATS 13*, Lecture Notes in Computer Science Vol. 8053, pp. 121-136, Springer-Verlag, 2013.

V Maryam Kamali, Luigia Petre, Kaisa Sere, and Masoud Daneshtalab. Refinement-Based Modeling of 3D NoCs, In F. Arbab and M. Sirjani (Eds.) *Proceedings of the 4th IPM International Conference on Fundamentals of Software Engineering - FSEN 11*, Lecture Notes in Computer Science Vol. 7141, pp. 236-252, Springer-Verlag, 2012.

VI Maryam Kamali, Luigia Petre, Kaisa Sere, and Masoud Daneshtalab. Formal Modeling of Multicast Communication in 3D NoCs. In P. Kitsos and S. Niar (Eds.) *Proceedings of the 14th Euromicro Conference on Digital System Design - DSD 2011*, pp. 634-642. IEEE/Euromicro, August 2011.

VII Maryam Kamali, Luigia Petre, Kaisa Sere, and Masoud Daneshtalab. CorreComm: A Formal Hierarchical Framework for Communication Designs, In *Proceedings of the 2nd IEEE International Conference on Networked Embedded Systems for Enterprise Applications - NESEA2011*, pp. 1-7. IEEE Computer Society, December, 2011.

VIII Maryam Kamali, Luigia Petre, and Kaisa Sere. NetCorre: A Hierarchical Framework and Theory for Network Design (*Submitted to Science of Computer Programming Journal*), April, 2013.

# Contents

**Part I: Research Summary** **1**

**1 Introduction** **3**

**2 Networking Architectures** **5**
2.1 Dynamic Networks . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.2 Static Networks . . . . . . . . . . . . . . . . . . . . . . . . . . 8

**3 Formal Methods** **11**
3.1 Event-B . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
3.2 Statistical Model Checking in Uppaal . . . . . . . . . . . . . . 16

**4 Reusable Formal Network Architectures** **19**
4.1 Reusable and correct-by-construction WSANs . . . . . . . . . 19
4.2 Towards a reusable implementation of WSANs . . . . . . . . 20
4.3 A reusable mobility model for analysing ad-hoc networks . . . 20
4.4 Quantitative analysis of routing in ad-hoc networks . . . . . . 21
4.5 Reusable and correct-by-construction NoC architectures . . . 21
4.6 Reusable and correct-by-construction multicast routing in NoC architectures . . . . . . . . . . . . . . . . . . . . . . . . . . 22
4.7 A correct-by-construction framework for developing static network architectures . . . . . . . . . . . . . . . . . . . . . . . . 22
4.8 A reusable and correct-by-construction network theory . . . . 23

**5 Related Approaches** **25**
5.1 Modelling and Analysing Dynamic Networks . . . . . . . . . . 25
5.2 Modelling and Analysing Static Networks . . . . . . . . . . . 26
5.3 Architectural Development of Networked Systems . . . . . . . 27
5.4 Correct-by-Construction System Development . . . . . . . . . 29

**6 Discussion** **31**

**7 Bibliography** **35**

**Complete List of Publications** **43**

**Pat II: Original Publications** **45**

# Part I

# Research Summary

# 1. Introduction

In recent years, novel networked systems have increasingly emerged at both large and small scale, such as wireless sensor networks, mobile ad-hoc networks, networks-on-chip, etc. These networked systems are uniquely designed and implemented, according to their specific principles. For instance, the design of wireless sensor networks is derived mostly by considering the power consumption, the data-centric nature of the sensor network, the node coordination techniques and the node failures. The design focus in mobile ad-hoc networks is on mobility, spatial correlation, in-network data mining and link failures. Networks-on-chip are designed to address on-chip communication issues and to provide for more efficient embedded systems.

The proliferation of these and other novel network paradigms leads to rethinking how to develop them in an efficient as well as reliable way. Efficient development allows one to manage the complexity of networked systems and shorten their development time. Producing reliable systems means that we can trust the developed system to behave according to its specification.

The overall goal of this thesis consists in contributing to the development of networking architectures for *efficiency* and *reliability*. This is a very large and diverse area of research whose impact can be enormous, due to the ubiquity of networked systems in our society. With respect to developing efficient network architectures, we focus on the *reusability* of our proposed models. With respect to developing reliable network architectures, we study the development of *correct models* from their specifications.

Our research method consists in applying formal methods toward achieving our research goal. Formal methods aim at increasing the rigour of the design and of the development of systems by employing mathematical-based techniques used for specification, development and verification of system models. Ultimately, formal methods aim at establishing software and system development as a discipline comparable to other engineering disciplines such as car manufacturing, avionics, construction and architecture, etc.

Networks can be classified with respect to several criteria, such as the medium used for transporting data, the scale, the topology, etc. With respect to the medium used for communication, we distinguish between *wired networks*, where waves are transmitted along a physical medium such as a

twisted pair cable or an optical fiber and *wireless networks*, where microwave, radio or infrared waves are propagated through air. With respect to scale, we distinguish between *local area networks* (LAN), that connect devices over a relatively short distance and *wide area networks* (WAN) that span a large physical distance. Network topology can be defined as the physical layout of the interconnections and the devices in a network. Such a topology can be *dynamic* when the layout evolves. A topology can also be *static* when the layout is fixed and unchangeable; in this case, we can distinguish between bus, star, ring and mesh topologies. In our research we focus on the topology taxonomy of networked systems, studying both static and dynamic networks. One reason for this choice is that it underlies both their functional and non-functional properties.

Thus, we propose reusable formal architectures for networked systems that can be applied to both dynamic and static networks. We evaluate our proposal by employing the proposed architectures to a pervasive class of dynamic networks, i.e., wireless sensor network architectures as well as to a pervasive class of static networks, i.e., network-on-chip architectures.

A complementary aspect of our work consists in developing systematic approaches for both qualitative and quantitative analysis of these networks. We employ two main modelling and analysis frameworks: a theorem prover and a statistical model checker. The study focus in qualitative analysis is on designing correctly functioning networks, while in quantitative analysis the study focus is on evaluating non-functional network properties such as performance.

Our research is based on a collection of eight papers. We split this dissertation in two parts. In Part I we present the context and overall view of the work, while in Part II we reprint the research papers (with permission). In Part I we proceed as follows. In Section 2, we describe our object of study, namely, our approach to networking architectures. In Section 3, we motivate and overview the research methods we employ. In Section 4, we put everything together and explain how we apply the methods described in Section 3 to the architectures described in Section 2; paper by paper, we characterise our contribution to the reusability and correctness of the developed networked architectures, emphasizing the qualitative or quantitative analysis that we perform. In Section 5, we discuss related approaches to this dissertation and in Section 6 we summarise our work as well as emphasize future research directions.

# 2. Networking Architectures

In this section we describe the design space of networked systems that is covered in our study. We point out the main aspects of networks that are taken into account in our architectural development.

A network consists of a set of interconnected devices that communicate with each other via messages. Three central features are at the core of every networked system: the network topology, the resource management and the routing algorithm. In this dissertation we refer to a *networking architecture* as a set of these three features.

The *network topology* determines the physical layout of the devices and of their interconnections in the network. A network topology can be defined as a graph $G = (V, E)$. The set of vertices $V = \{v_1, v_2, ..., v_n\}$ models the network devices, called *nodes*. The set of edges $E = \{e_1, e_2, ...., e_m\}$ models the interconnections, called *links* between the nodes; an edge $e_i$, $i \in \{1, ..., m\}$ is expressed as a pair of vertices $(v_j, v_k)$, where $v_j, v_k \in V$. The *resource management* refers to the usage of the edges and vertices of $G$ for the functioning of the network. A central concept for network functioning is that of a *message*. A message is a communication unit in a network and is typically characterized by a source node, one or more destination nodes, as well as some data. A *routing algorithm* determines the paths (also called routes) through the network, that messages take to reach their destinations. Routing algorithms can be classified into several categories, for instance adaptive or deterministic and unicast or multicast routing. In an adaptive routing algorithm, a path from a particular source to a destination is determined when it is required, by considering the network state. In a deterministic routing algorithm, the routes of the messages to destinations are determined at the initialization of the network; messages are always routed on a fixed path between a particular source and destination. Adaptive routing algorithms are preferable in networks with frequent topology changes and non-uniform network traffic: a better route might always be found in a new configuration. Deterministic routing algorithms are efficient in networks with relatively stable topologies and regular traffic patterns: they can safely reuse already established routes. A unicast or multicast routing algorithm refers to the message delivery scheme in networked systems. In the unicast

category, a message is delivered to a single specific destination node, while in the multicast category, the same message is sent to a set of destination nodes.

In this dissertation, our goal is to model, develop, and analyse reliable networking architectures for reusability. Our starting point consists of a very abstract representation of a network topology - the graph $G$ above -, that is associated with abstract resource management and routing. Already in this abstract view, two networking architectures can be derived: (i) a network model with a varying set of nodes and interconnections and (ii) a network model with a fixed set of nodes and interconnections. We refer to the models (i) for networking architectures as *dynamic networks* and to the models (ii) for networking architectures as *static networks*.

Modelling, developing and analysing networking architectures is highly influenced by the network type. For this reason, in the following we put forward the most important issues concerning topology, resource management, and routing algorithms with respect to dynamic and static networks.

## 2.1 Dynamic Networks

The main concern in a dynamic network is to achieve its desired functionality while the network topology frequently changes. The network topology changes due to node or link failures. The node failure can happen at any time due to unpredictable changes in the network, such as malfunctioning, shortage of power, etc. The link failures usually happen when nodes are mobile and move out of the range of their neighbours.

**Network topology** According to the two types of failures, we study two essential features in our dynamic network topologies: *unpredictable changes* and *mobility*. The dynamic topology keeps changing with time and thus, we can consider the network goes through a series of transition points $t_i$, for $i > 0$. A transition point $t_i$ corresponds to a particular moment when the network topology changes from $G_i = (V_i, E_i)$ to $G_{i+1} = (V_{i+1}, E_{i+1})$.

We specify the behaviour of the network topology in the presence of unpredictable changes in the network with the following three sets of changes for a transition point $t_i$:

$$add_edges_{(i)} = \begin{cases} V_{i+1} = V_i \\ E_{i+1} = E_i \cup \{\alpha\}, \quad \alpha \notin E_i \end{cases}$$

$$add_vertices_{(i)} = \begin{cases} V_{i+1} = V_i \cup \{m\}, & m \notin V_i \\ E_{i+1} = E_i \end{cases}$$

$$delete_vertices_{(i)} = \begin{cases} V_{i+1} = V_i \setminus \{m\}, & m \in V_i \\ E_{i+1} = E_i \setminus \{\gamma\}, & \gamma = \{(a,b) \mid a = m \vee b = m\} \end{cases}$$

These changes model the following situations. When a new link between two neighbour nodes modelled by vertices $m$ and $k$ is discovered in the network, the corresponding edge $\alpha = (m, k)$ is added to the set of edges and the set of vertices does not change ($add_edges_{(i)}$). Nodes can be added at any time to a dynamic network, leading to a change in the set of vertices of the network graph, as described by $add_vertices_{(i)}$. When a node (e.g., modelled by vertex $m$) fails, its corresponding vertex is deleted from the set of vertices. As a result, all its edges $\gamma$ are also deleted from the set of edges ($delete_vertices_{(i)}$).

We specify the behaviour of the network topology in the presence of mobility with the following two sets of changes for a transition point $t_i$:

$$add_edges_{(i)} = \begin{cases} V_{i+1} = V_i \\ E_{i+1} = E_i \cup \{\alpha\}, & \alpha \notin E_i \end{cases}$$

$$delete_edges_{(i)} = \begin{cases} V_{i+1} = V_i \\ E_{i+1} = E_i \setminus \{\alpha\}, & \alpha \in E_i \end{cases}$$

When a mobile node modelled by say, vertex $m$, enters the transmission range of another node, modelled by say, vertex $k$, the edge $\alpha = (m, k)$ corresponding to a link between vertices $m$ and $k$ is added to the set of edges of the network graph; the set of vertices does not change ($add_edges_{(i)}$). When a mobile node modelled by say, vertex $m$, leaves the range of one of its neighbours modelled by say, vertex $k$, the edge $\alpha = (m, k)$ is deleted from the set of edges and the set of vertices does not change ($delete_edges_{(i)}$).

**Resource management** In dynamic networks, nodes are untethered autonomous devices with power limitations due to their device types. Various device types result in correspondingly various resources with their specific computation abilities and communication ranges. For instance, some more powerful nodes may route messages from more limited nodes and some nodes may act as gateways to long-range data communication networks. The degree of heterogeneity in dynamic networked systems is an important factor affecting the resource management.

To address resource management features in dynamic networks, we have studied two different types of nodes together with their relationships. Considering heterogeneity in a networked system design provides the means to model and analyse connectivity at different levels. We have specified the behaviour of the network at three different levels, based on three connectivity graphs. Two of these graphs model the homogeneous interconnections, between nodes of the same type. The third graph models the heterogeneous interconnections, between nodes of different types.

**Routing algorithm** In order to discover routes in dynamic networks, the network should be connected, i.e., there should always be a path, possibly over multiple hops, between any two nodes. However, due to topology changes, the network may transform into a set of sub-networks disconnected from each other; this set is called a network *partition*. In particular, this means that there can be nodes in the network that have no connection and messages cannot be transported between them. A recovery mechanism is necessary in this case, to re-establish connections between the sub-networks. The recovery should guarantee the establishment of connection between the sub-networks. We address the connectivity issue in our dynamic network architecture by discovering a sub-graph of the network topology graph denoting alternative links between separated sub-networks.

Apart from connectivity, route discovery and maintenance play a crucial role in the reliability and the performance of dynamic networks. The Ad Hoc On-Demand Distance Vector Routing protocol (AODV) [56] is an efficient routing algorithm with different variants, aimed at improving the performance of dynamic networks. To reason about AODV and its variants in dynamic networks, we use a dynamic topology model that considers mobility.

## 2.2 Static Networks

Static networks have a fixed set of nodes and interconnections. The main concern in a static network is to function predictably, i.e., according to a given specification. Hence, one important area of study is how to design and reason about various given specifications for network functioning.

**Network topology** The main characteristic of a static network consists in its predictability. With respect to the network topology, this means that we can specify and *reuse* various topologies. In particular, we can develop a hierarchy of static network topologies, where the most abstract one defines concepts common to all network topologies. Such abstract topology is essentially made of the graph $G = (V, E)$ with the vertices $V$ and the

corresponding edges $E$ between vertices not evolving in time. We then specialize this graph into a very common topology for static networks, namely the *mesh topology*. The mesh topology is a regular graph structure that makes analysis of static networks possible. In a mesh topology, nodes can be laid out to form a 2-dimensional (2D) rectangular grid in which each node is connected to its four neighbours except for the nodes of the boundary. We specify the topology graph of a 2D mesh network for a fixed number of nodes, say $\mid V \mid = N$. In such a mesh topology, we have a $\sqrt{N} \times \sqrt{N}$ mesh structure, and the number of edges is less or equal than $2N$: $\mid E \mid \leq 2N$. The connectivity degree $d_u$ of a node $u$ stores the number of pairs $(v, u) \in E$, for any $v \in V$; $d_u$ is also an important factor in the topology of a static network. For a 2D mesh network, we have that $d_u \in [2, 4]$ for any $u \in V$. We can of course specialize the abstract topology to a n-dimensional (nD) mesh topology. For $n = 3$ (a 3-dimensional (3D) mesh topology) a fixed set of nodes $N$ ($\mid V \mid = N$), we have a $\sqrt{N} \times \sqrt{N} \times \sqrt{N}$ mesh structure with $\mid E \mid \leq 3N$ and the connectivity degree $d_u$ is so that $d_u \in [3, 6]$ for any $u \in V$.

**Resource management** To analyse a systematic and effective architecture for static networks, apart from the *message* resource, other resources such as *channels* and *buffers* should be managed. A message, which is generated by a node, is further divided into *packets* and a packet can be further split into a number of *flits*. A channel is a link that connects nodes together. In fact, channels are instantiations of *edges* in the network topology, allowing to model more network characteristics. A buffer stores messages temporarily in the start and end nodes of the channels. Moreover, in order to manage allocating buffers and channels to messages, we need to define the *switching* mechanism that determines how the message moves along the network path. Regarding the switching mechanism, buffers and channels can be allocated to either messages, packets or flits. In message-based switching, channels and buffers between a source and a destination across multiple hops are allocated to the whole message. In packet-based switching, the allocation of channels and buffers is handled independently for each packet. In flit-based switching, also called *wormhole*, flits of a packet follow each other and the same set of channels and buffers are allocated to each individual flit of a packet, at different moments.

We model and analyse the channel and buffer structural elements of a static network as well as messages, packets and flits in our architecture for static networks. We also address switching techniques, to manage the allocation of channels and buffers at the message, packet and flit level.

**Routing algorithm** In static networks the issues in designing routing algorithms are to provide high throughput and low latency, and also to be

deadlock free. There are many routing algorithms that are proposed for various topologies of static networks ranging from adaptive to deterministic routing schemes, to provide efficiency in the presence of different congestion conditions. In adaptive routing techniques in static networks, the route decision is taken in each node, with respect to network traffic. In deterministic routing techniques, the minimal route is typically chosen. As the network topology does not change, nodes always route messages on the same path. Therefore, the behaviour of nodes is different in a static network with adaptive or deterministic routing algorithms. Moreover, in networks with unicast routing technique, nodes only need to route a message to one outgoing channel. In networks with a multicast routing algorithm, nodes need to take decisions about routing a message on one or more outgoing channels. In order to address different characteristics of routing algorithms at the architectural level, we model a hierarchy of fundamental architectural components and analyse them at different levels of abstraction.

# 3. Formal Methods

In this section we describe the formal tools that we employ throughout the thesis. Formal methods can be seen nowadays as a contribution of computer scientists to raise software and software-intensive system development to an engineering level comparable to well established disciplines such as car manufacturing, avionics, construction and architecture, etc.

One essential artefact common to these engineering disciplines is that of *blueprint* or *model*. A model allows us to address the complexity of the system under development by *abstraction*, i.e., illustrating only the features of interest with respect to a particular model's purpose.

In model-driven development, first a high-level model of a system is developed (often via numerous steps) and then the model is developed into a specification closer to the implementation of the system. This development can consist of adding details and functionalities and removing non-determinism and is typically referred to as *refinement* [20]. The concepts of *abstraction* and *refinement* are at the core of such a stepwise development. Abstraction is an essential feature for reusability. The abstract specification typically describes *what* the system does, whereas the more concrete specifications derived by refinement describe *how* it is done. In fact, the architecture of a system is introduced in a high-level specification and the design decisions are gradually introduced in low-level specifications.

Formal methods are broadly divided into state-based and event-based, corresponding to the first class entities in the models. State-based methods deal with specifying systems by variables and operations that modify these variables. The values of the variables describe the *state* of the system. Action Systems [16], the Z notation [63], the B-method [9], VDM [53] and Event-B [10] are all examples of state-based formal methods. Event-based formal methods focus on representing systems by a composition of processes that communicate via channels. Well-known examples of event-based methods are CSP [39], CCS [49] and $\pi$-calculus [50]. A combination of state-based and event-based methods are also proposed as CSP||B [59] and CSP||Event-B [60].

Nowadays, tool support is an essential instrument for the usability of a formal method. One reason for this is that tools provide a platform to cope

with system modelling and analysis at the same time; this promotes the efficient use of formal methods. Another reason is that tools have features for syntax checking and also, at some levels, they can automatically prove trivial properties of a desired system. According to their type, tools associated to formal methods can be theorem provers, model checkers and simulators. In this dissertation, we employ the state-based formal method *Event-B* that has an associated theorem prover tool, namely the Rodin platform [7, 12]. We also find instrumental the event-based formalism of *timed-automata* [13], supported by tools such as the Uppaal [21] model checker and its statistical extension called SMC-Uppaal [28].

Common to all formal methods is the concept of a *precise* (or *formal*) *model*. This essentially means that we can associate a meaning to the model, called *semantics*. Semantics can be defined as a mapping from a novel concept to a concept that we already understand well. Capturing a system at several levels of abstraction via formal models allows for two types of activities that are specific to formal methods: formal development and analysis of various properties.

Event-B combines the idea of model-driven development and formal methods to construct a correct model of a system. In this framework, a formal abstract model of a system with its desired properties is specified and proved correct with respect to its specification. The high-level specification is then developed at several different refinement levels. The notion of refinement is mathematically defined in Event-B in a way that allows us to analyse the relationship between formal specifications at different levels of abstraction. This corresponds to a *qualitative* analysis of the system.

In Event-B, in each refinement step, certain design decisions are introduced into the system specification. There might be various design alternatives that can provide the desired functionality with distinct consequences for the non-functional properties of the system. In order to choose the optimal design alternative, a *quantitative* analysis of non-functional properties, (e.g., performance) should also be undertaken.

To assess system performance in a networked system, the probability that a system correctly functions over a given period of time should be evaluated. To achieve quantitative assessment in networked systems, we use an extension of timed-automata in Uppaal for system specification, called *price timed-automata*; this integrates the notion of time and probability into classical timed-automata and uses a statistical model to verify the quantitative properties of networked systems.

In the following we briefly overview the formal frameworks that are employed in this dissertation for both qualitative and quantitative analysis of networked systems.

## 3.1 Event-B

Event-B [10] is a formal approach for the specification and development of highly dependable distributed systems. Event-B comes along with the associated tool Rodin [12] which provides a platform for specifying and verifying distributed systems based on a theorem prover.

Event-B allows us to formally model a system and to prove that the model fulfils certain desired properties. A system is simulated by constructing models which will be analysed by doing proofs. To perform simulation and proofs, a discrete transition system formalism is used. The system simulation is represented by means of a succession of *states* with *transitions*, called *events*. The proof is performed by demonstrating that the transitions preserve a number of desired global properties which must be guaranteed by the states of the system components. These properties are called *invariants*. A system state is defined by a set of variables and constants at a certain level of abstraction and changes by taking a transition that can occur under certain circumstances. Transitions are made of a guard and an action. A guard is the necessary condition under which an event can occur and is expressed as a predicate on the state constants and variables. An action introduces the way in which state variables are changed as a consequence of the event occurrence. In order to be able to reason about a system model, we construct closed models: the interaction between an environment and a corresponding controller is clearly regulated and preserved. In fact, we need to model the controller, an abstract environment within which the system behaves, as well as their interaction. Therefore, the large number of states and transitions of such systems are categorized into environment and controller. To master the complexity of such real systems with large number of states and transitions, Event-B employs the concept of refinement, developing a system model by a number of correctness preserving steps.

**Refinement** The refinement concept was first introduced by Dijkstra [33] and Wirth [71] for developing correct programs. The logical foundation of refinement based on the weakest precondition approach [34] was later developed into the refinement calculus framework by Back and Von Wright [32]. The refinement concept provides for a top-down approach to constructing systems following rules for gradually introducing details to an initial abstract specification. In stepwise refinement development, an abstract, nondeterministic specification is transformed into a more concrete and deterministic one in such a way that the correctness is preserved during the transformation. In fact, the refinement relation guarantees that each intermediate refined specification from $R_1$ to $R_N$ as well as the *Executable program* are correct refinements of the initial abstract specification ($R_0$), as illustrated below. Therefore, the system developed by refinement is correct-by-

construction:

$$R_0 \sqsubseteq R_1 \sqsubseteq \ldots R_N \sqsubseteq \textit{Executable program}$$

There are two types of refinement: *horizontal* refinement and *vertical* refinement. In horizontal refinement (superposition refinement) [18], a large system is modelled in successive steps so that the model becomes richer in each step by first creating and then enriching states and transitions of the model components. In vertical refinement (data refinement) [15, 19], the task is to transform some states and transitions of the model to more concrete states and transitions that are easier to implement.

In our networked system development in Event-B, refinement is the central method by which initially abstract models of networked system features are developed through detailed design toward code. At the abstract level, the architectural components of networked systems are introduced and the global properties of such systems are analysed. This abstraction is beneficial for constructing correct reusable models. On one hand, abstraction allows us to generalize the specification of a set of systems with common behaviours. On the other hand, it allows us to prove the global properties of such general specifications. Then the successive steps of refinement allow to unfold components of the abstract model one-by-one while the global correctness of our networked system is preserved. In addition, the refined model is reusable in the development of different types of networked systems. This approach provides means to construct reusable generic models while preserving the global correctness of the system.

Apart from *refinement*, Event-B benefits from *pattern-driven formal development*, *decomposition* (modularization) and *theory extension* features to provide for correctness-by-construction and reusability in system development.

**Pattern-driven formal development** This approach aims to identify general model transformation solutions called *refinement patterns* and then repeatedly apply them to facilitate the refinement process. Refinement patterns are used in pattern-driven formal development to provide a basis for automation and reusability. Pattern-driven formal development can be represented as shown in Fig. 3.1. The initial model $M_1$, the starting point of the development, is created by instantiating a special template, called *specification pattern*; this is a parametrised specification. During pattern instantiation, the model parameters are substituted with concrete data structures, while the model variables and events can be renamed. The model constraints given for these parameters become the theorems to be proved in order to show that this pattern is applicable for the given concrete values. If the instantiation succeeds, the model invariant properties (together with their proofs) are obtained for free, i.e., without any proofs.

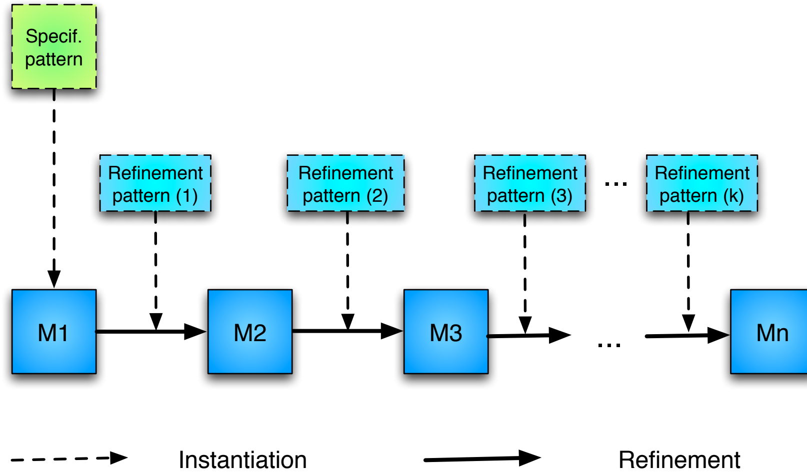



Figure 3.1: Pattern-driven model development

**Decomposition** The complexity of system models can be addressed by stepwise refinement development. However, after several refinement steps, the system state may drastically increase. The growing state size and number of transitions in a single model is a barrier for scalability. Therefore, it is necessary to break a single model into several component models in a systematic fashion, i.e., by decomposition. By decomposing a system into a set of component models, the complexity management of the whole system reduces to studying and refining each component model independently of the others. Apart from mastering the model complexity, decomposition facilitates scalability and reusability of component models [7].

Modularization in Event-B is a form of decomposition where *modules*, components containing groups of callable operations, can be developed separately and then composed with the main system during its formal development [43]. The modularization in Event-B is based on the idea of *monotonicity* [17]. This means that, if a sub-component $M_i$ of an abstract system model $M$ is correctly refined by $M_i'$, then the model $M$ where the component $M_i$ is substituted by $M_i'$ must be a correct refinement of the abstract model $M$. A module is built of a model interface and a model body. A module interface defines module operations and the way in which the module external variables may change between operation calls. A module interface is then implemented by a module body which provides a concrete behaviour for interface operations [42].

**Theory Extension** Event-B also provides a feature for extending its standard mathematical language by supporting user-defined operators and predicates [29, 11]. Moreover, it also allows us to define new proof rules along

with these notations. Both additional notations and proof rules are defined as a *theory* component that can be then used in future Event-B models. A theory can consist in new algebraic types, new operators and new proof rules and their validation is ensured by proving the generated proof obligations. The theory feature enables the definition of reusable types, operators as well as proof rules as a library of theories that can be employed in Event-B models and development processes.

## 3.2 Statistical Model Checking in Uppaal

Numerical model checking accurately computes the probability that a system satisfies a temporal logic property. However, the applicability of numerical model checking is generally limited to systems with a small number of states. The state-number restriction in model checking does not allow to verify large models such as protocols in large networks or under different conditions. Therefore, for the verification of *quantitative properties* of network protocols, we need to employ other methods, such as statistical model checking (SMC). SMC [73, 61] combines ideas of model checking and simulation with the aim of supporting quantitative analysis as well as addressing the size barrier that prevents useful analysis of large models.

Statistical model checking addresses the verification problem of large systems by providing a statistical evidence for the satisfaction or violation of the specification [73, 61]. It computes the probability that a system model satisfies a system property. For instance, in protocol verification, it can compute the probability of route discovery in the presence of mobile nodes in the network. This problem cannot be solved by numerical model checking techniques due to the state explosion problem. Thus, the idea of this approach is to conduct some simulation of the system and to verify if this simulation satisfies a given property.

SMC is based on a formal semantic of systems and uses Monte Carlo sampling and hypothesis testing to reason on behavioural properties of systems. In fact, a system is simulated for many runs. The number of simulation runs needed in SMC-Uppaal is computed by using Chernoff-Hoeffding bounds, that are independent of the size of the system. Extracted samples are monitored by testing techniques and a statistical evidence for the correctness of the system with respect to the system properties is provided. The statistical evidence is an accurate estimation, because it is computed on sample runs of the system according the distribution defined by the system. Therefore the probability measures follow the same distribution as defined by the system.

SMC-Uppaal supports the analysis of price timed-automata (PTAs). PTAs are timed-automata in which clocks may have different rates in different locations. PTAs are input-enabled, deterministic automata that com-

municate via broadcast channels and shared variables to build networks of price timed-automata (NPTA). Properties of NPTAs are expressed in cost-constraint temporal logic [31, 64] over runs of the form $\psi = \Diamond_{x \leq c} \varphi$, where $\varphi$ is a state-predicate, $x$ is an observer clock which never resets and $c \in \mathbb{R}_{\geq 0}$ is a bound. Each run is encoded as a Bernouli random variable that is true if the run satisfies $\varphi$ with $x \leq c$ and false otherwise. For an NPTA $M$, $P_M(\psi)$ is the probability that a random run of $M$ satisfies $\varphi$.

To answer the problem of checking $P_M(\psi) \geq p$ ($p \in [0, 1]$), statistical model checking algorithms are used. We can have two types of analysis: qualitative and quantitative. In qualitative analysis, the goal is to answer if the probability $P_M(\Diamond_{x \leq c} \varphi)$ related to a given NPTA $M$ is greater or equal than a certain threshold $\theta$:

$$P_M(\Diamond_{x \leq c} \varphi) \geq \theta$$

In quantitative analysis, the goal is to find the probability that a random run of a given NPTA $M$ satisfies $\Diamond_{x \leq c} \varphi$. The qualitative analysis issue in SMC-Uppaal is addressed by the hypothesis testing approach. In this approach, the hypothesis $H : p = P_M(\Diamond_{x \leq c} \varphi) \geq \theta$ is tested against $K : p < \theta$. The quantitative analysis issue in SMC-Uppaal is addressed by the Monte Carlo approach. Namely, the number of runs needed to produce an approximation interval $[p - \varepsilon, p + \varepsilon]$, where $p$ denotes the probability of $\psi$ with a confidence $1 - \alpha$ is computed. Here, $\alpha$ stands for the probability of false negatives and $\varepsilon$ stands for the probabilistic uncertainty. These parameters are used to specify the statistical confidence on the result.

# 4. Reusable Formal Network Architectures

In this section we put forward the contribution of the original publications that make up this dissertation. In particular, we discuss the approach taken in each paper to address the *reuse* and *correctness* of networking architectures. Moreover, we emphasize the quantitative and quantitative analysis aspects put forward by each paper.

## 4.1 Reusable and correct-by-construction WSANs

In this paper, we model and analyse the functioning of a dynamic network in Event-B. Based on this, we then derive design patterns for developing reliable dynamic networks. With respect to the three components of a networking architecture (i.e., topology, resource management and routing), in this paper we address the following. The network topology changes due to unpredictable failures of nodes. The resource management feature is studied by considering two types of nodes with their corresponding homogeneous and heterogeneous connections. Routing is addressed by modelling the connectivity aspects of a dynamic network upon the failure of a node. A formal architecture for developing dynamic networks is thus proposed, including three main features: unpredictable topology, heterogeneous nodes and connectivity. These features are developed in several refinement steps. The stepwise methodology reduces the proofs of the system *correctness* at each level, because in each refinement step one feature is detailed. Dealing with one feature at each step also provides for a compositional approach. The developed dynamic network is analysed against its desirable mathematical properties. The main property we verify is the satisfiability of reestablishing connectivity in a dynamic network; this is a rather thorough example of qualitative analysis. We put forward the *reusability* of our methodology via formal design patterns. We present an abstract model of our architecture together with possible refinement patterns that can be reused to develop and analyse a particular dynamic network. Our example of a dynamic network is a wireless sensor-actor network (WSAN).

## 4.2 Towards a reusable implementation of WSANs

In this paper we start from the same model of a networking architecture as in Paper I. Namely, we study a dynamic network whose topology changes due to unpredictable failures and whose resource management is addressed again via two types of nodes. However, in Paper II we take a different view on the development and *reusability* feature of our network. In particular, we discuss the derivation of a dynamic network *implementation* from part of the abstract specification presented in Paper I. In the proposed architecture of Paper I, the network features are compacted in an individual system model: this makes it challenging to derive an implementation of a dynamic network from the specification. To tackle this and provide a more efficient development approach, we employ the decomposition technique in Event-B. More precisely, we develop a model for a network infrastructure and a model for the distributed nodes, using the modularization technique in Event-B. We define node modules which have their own state and invariant properties. The *correctness* of the decomposed model follows from the proof obligations that express the consistency between the abstract and the refined model; again, we put forward a qualitative analysis for our proposed architecture. By decomposing the model of a dynamic network, we provide means to reuse the node modules and the network infrastructure to develop specific dynamic network architectures, which can then be efficiently implemented.

## 4.3 A reusable mobility model for analysing ad-hoc networks

In this paper, we focus on modelling a dynamic network, whose topology evolves due to node mobility. The resource management and the routing features are considered as general as possible, and hence, have no explicit definition in this paper. Our aim here is to efficiently model node mobility, in order to evaluate the performance of network protocols by simulations. We propose an abstract, reusable, automata-based model that defines mobility as probabilistic changes in the topology. The model is instantiated to two specific mobility models, namely random walk and random waypoint, to express *reusability* of the automata-based model. The proposed mobility model provides a sound foundation for reasoning about the behaviour of dynamic networks such as mobile ad-hoc networks. In other words, the model is intended to be used in conjunction with protocols for efficient performance analysis of dynamic network protocols and thus provides the basis for a quantitative analysis of the proposed architecture.

## 4.4 Quantitative analysis of routing in ad-hoc networks

In this paper, we focus on the quantitative analysis of the dynamic network model proposed in Paper III. We analyse several variants of the AODV routing protocol and compare them, to determine optimal versions. We model a dynamic network where the topology changes are due to node mobility. The resource management feature is addressed by considering two types of nodes in the network: static and mobile. The routing feature is studied by modelling the AODV routing protocol and its variants. We develop a mobility model as a price timed-automata. AODV and its variants are specified using also priced timed-automata. The probability of route discovery is analysed in all AODV variants to determine which variant performs better with respect to route discovery. The mobility model is an independent time automata that can be *reused* for analysis of network protocols in networks where the topology changes due to mobility. The mobility model is a parametrized template which can be instantiated to provide different patterns for the changing topology.

## 4.5 Reusable and correct-by-construction NoC architectures

In this paper, we propose a formal architecture for the efficient development of static networks. In particular, we develop a network-on-chip architecture using Event-B. We address the modelling of a static topology and focus on the 3D mesh topology. The resource management feature is modelled by considering network resources such as packets, channels and buffers. By considering the packet as a resource, we focus our development on packet-based switching mechanisms. The routing feature is addressed by studying unicast communication. These network features are developed in three refinement steps. The initial abstract model includes a static topology that is independent of the real physical layout, a set of messages and the unicast communication functionality. In the first refinement steps the mesh topology and the concept of channels are added and in the second refinement step the concept of buffer is added. Already in the first abstract model, the sufficient condition for constructing a *correct* network architecture is defined and proved. The sufficient condition is that all injected messages are received by their destinations. In the next refinement steps the condition is developed to take into account the specific features of these steps. In each refinement step, the proving process guides us to develop a correct architecture. The refinement decisions are taken so that each of the three models can be reused to develop and analyse specific network designs at different abstraction levels.

We demonstrate the *reuse* of our development, by modelling and analysing a particular routing algorithm, as a refinement of our third model. Thus in this paper we put forward a qualitative analysis of the proposed network architecture.

## 4.6 Reusable and correct-by-construction multicast routing in NoC architectures

In this paper, we propose a reusable architecture for static networks with a multicast communication scheme. We start from the same topology and resource management models of a networking architecture as in Paper V; however, multicast communication is taken into account for modelling the routing feature. The network development is performed in three refinement steps. In the initial model, structural components and sufficient conditions for modelling and analysing multicast communication are proposed. In the next two refinement steps, the structures are refined to add the concept of channel and buffer. The sufficient condition is that all injected messages are received by all their destinations; this condition is preserved by our development. The generic models specify the primitive structures and sufficient conditions for constructing a correct network, regardless of any specific network architecture. The generic aspect of the proposed development is key for *reusability* and provides another example of qualitative analysis of static network architectures.

## 4.7 A correct-by-construction framework for developing static network architectures

In this paper, we propose a hierarchical framework for the development of correct static network systems. We combine models of static networks proposed in Paper V and Paper VI for developing network-on-chip systems, to gain a more general model that can be reused for a wider class of static networks. The framework addresses the modelling of a static network regardless of its physical layout. To study the resource management feature, we consider packets, flits, channels and buffers. By considering both packets and flits, our framework becomes suitable for modelling both packet-based and flit-based switching mechanisms. The routing feature is addressed by considering both unicast and multicast communication. The framework provides a clear separation between architectural and algorithmic aspects of network development and, as a consequence, provides means for easier discovering error resources in the development cycle. The framework is constructed in Event-B and is an abstract and parametric description of static networks;

its *correctness* is satisfied by proving the satisfaction of sufficient conditions. The strength of our framework consists in *reusability*. The pre-proved framework provides a reusable network infrastructure that allows designers to develop specific network architectures correctly by proving all proof obligations in their development. It also provides the means to construct a library of verified components to be employed for the development and qualitative analysis of a specific network design.

## 4.8 A reusable and correct-by-construction network theory

In this paper we strengthen the framework proposed in Paper VII by amending a set of basic formal theories for developing network architectures at an abstract level. The purpose of this extension is to use lessons learned from previous architectural developments to address our ultimate aim, i.e., providing a network theory that can be reused for the development of correct networked systems. We address the modelling of a generic topology that can represent both static and dynamic topologies. The resource management feature is modelled by considering a message, as a general concept that can be instantiated to packets and flits; we also consider channels and buffers. The routing feature is addressed by studying general communication schemes that can be seen as unicast and multicast or adaptive and deterministic. For these features we define and prove *correct* rules as theories in Event-B. As theories are general, they are *reusable* for the development of correct networked systems. In fact, we develop a library of pre-proved rules, representing the functionality of network features in a hierarchical manner. The pre-proved rules facilitate the design and verification of networked systems throughout the network development cycle. The developed rules can be *reused* as original constructors in Event-B that alleviate the difficulty of using formal methods for network development. Another advantage of using such rules is that the designer can freely develop networked systems for qualitative analysis of other features of networked systems without considering refinement decisions that have been taken in our previous works.

# 5. Related Approaches

In this chapter, we review existing literature that we find relevant to this dissertation. First, we discuss formal approaches for modelling and analysing dynamic networks. Second, we outline approaches for correctly developing static networks. Third, general approaches for specification and verification of network protocols without considering certain characteristics of dynamic/static networks are discussed. Finally, we discuss correct-by construction approaches for system development.

## 5.1 Modelling and Analysing Dynamic Networks

Formal modelling and analysis of dynamic networks has been widely addressed in the literature. Bernardeschi et al. [22] propose an approach based on the PVS theorem prover [6] to analyse protocols for sensor networks in dynamic scenarios with mobile nodes. They develop a formal specification for the reverse path forwarding (RPF) algorithm, which is a broadcast routing method. RPF exploits the information contained in the routing table to deliver packets generated by a base station to all other nodes in a multi-hop network. They use PVS to verify correctness properties of RPF. The similarity of their approach to ours consists in using abstraction and refinement techniques in network development.

Bhargavan et al. [23] use the theorem prover HOL [4] and the model checker SPIN [8] together to prove properties of routing protocols in ad-hoc networks. They verify the AODV routing protocol using their method and identify errors in the AODV specification that can lead to a deadlock situation. Fehnker et al. [36] propose a process algebra for wireless networks to verify properties of network protocols. They model AODV in this framework and derive a Uppaal model of AODV from their process algebra model to check desired properties of AODV against all topologies of up to 5 nodes [35]. This exhaustive search allows to quantify in how many topologies a particular error can occur. They also analyse the AODV model in a dynamic network when a link breaks. Continuing this line of research, Hoefner and McIver [40] use the SMC extension of Uppaal to verify properties of the AODV in larger networks, with up to 100 nodes. We extend this series of

studies on the AODV routing algorithm by adding a mobility model. We capture a dynamic network topology and analyse properties of AODV.

Xiong et al. [72] propose a timed model to verify routing protocols for wireless ad-hoc networks, based on the idea of topology approximation. This approach describes aggregate behaviour of nodes when their long term average behaviours are of interest. They use Colored Petri Nets (CPN) [44] to demonstrate the applicability of their approach by modelling and verifying AODV. Yuan et al. [74] model the dynamic topology changes of ad-hoc networks with Colored Petri Nets and verify a routing protocol for mobile networks called Destination-Sequenced Distance Vector (DSDV) [57] to exemplify their technique.

The purpose of the outlined studies consists in mostly specifying and verifying existing network protocols. Our approach focuses on the development of correct network protocols by investigating the genericity and reusability of proof-based models. We propose sufficient architectural conditions and structural design patterns and methods to design correct network protocols that satisfy the essential conditions of network architectures. Our approach would ensure that properties established on the abstract models are satisfied by the actual implementation.

## 5.2 Modelling and Analysing Static Networks

The literature on modelling and analysing static networks has grown in the recent years. Schmaltz and Borrione [58] propose a general model of on-chip communication architectures, called GeNoC, in order to facilitate the design of correct network-on-chip systems. Their model is developed using the ACL2 theorem prover [1]. Three independent groups of functions, namely routing and topology, scheduling, and interfaces, form the foundation of the model. Moreover, the sufficient constraints that these functions should satisfy are introduced in order to prove the correctness of the model. This separation of functions allows for a stepwise design and verification, where at each step only one group of functions is considered. We also adopt this feature in our framework. The GeNoC model is used to prove properties of communication functions for the routing algorithm of the HERMES NoC [26, 68] and of the spidergon NoC [27]. The purpose of the above studies is to verify the overall correctness of these NoC systems. Verbeek and Schmaltz [65, 66, 67] extend GeNoC to provide sufficient constraints that ensure deadlock-free routing and liveness of the design.

The genericity and reusability proposed by GeNoC are similar to our approach. However, we provide these features using the powerful technique of refinement, to manage the complexity of the development as well as of the proofs. Moreover, we have the possibility to derive an implementation of a

certain NoC architecture from a developed model, using refinement.

Andriamiarina et al. [14] have only recently proposed a formalism for NoC architectures based on incremental design and proof theory. They develop their specification using the Event-B formalism to verify an adaptive and fault-tolerant routing technique. They focus only on a particular type of routing technique in their development, which reduces the generalisation of the development.

Chen et al. [30] propose a formal modelling approach to verify routing protocols for NoC architectures. They provide a guideline for constructing formal models of NoC designs and propose a methodology for verifying NoC properties such as deadlock freedom and traffic congestion. For the formal verification task, a model checker called State Graph Manipulators (SGM) [41] is used and they show the applicability of their approach by developing and verifying a specific NoC, namely the Bidirectional Channel Network-on-Chip (BiNoC) [48]. However, due to applying model checking in verification, this work suffers from the state explosion problem. Therefore, the number of nodes is a concern in this verification while the number of nodes is not a limitation for our work.

Palaniveloo et al. [55] propose a formal model of the Hermes NoC router architecture and its communication scheme using their own Heterogeneous Protocol Automata (HPA) formalism. The HPA language is developed in this work to model the behaviour of communication modules as event-based transition systems. They also map the automata model of NoC developed in HPA to PROMELA specification language of the SPIN model checker [8] for verification.

Böhm [24] proposes a formal framework for modelling and verifying on-chip communication protocols in the Isabelle/HOL theorem prover [54, 5]. The proposed methodology is based on incremental modelling in which abstract building blocks and composition rules are initially specified. This relates to our work of using an incremental approach that interleaves model construction and verification. Some other refinement approaches to the design and verification of on-chip communication architectures have also been used in [25, 38].

## 5.3 Architectural Development of Networked Systems

The formal approaches discussed in Sections 5.1 and 5.2 entail the specification and verification of networks at a high and abstract level. The high-level specification should be transformed to an actual implementation while it ensures the correctness of its properties. The correct transformation of a high-level specification to an implementation is a challenge. To tackle this

challenge, there are a series of studies on the use of formal analysis techniques to reason about network protocol correctness throughout the network development cycle [45, 37]. They mostly propose domain specific languages in order to develop correct network protocols.

Karsten [45] axiomatically specifies basic inter-networking concepts. This is then employed to construct a theoretically sound framework, to express architectural invariants and the deliverability of messages even in the presence of network dynamism. A meta-language is proposed for the rapid implementation of different packet forwarding schemes. The concepts and the meta-language derived from them aim at clarifying the essential architecture of the Internet. Moreover, they provide a bridge between formal proofs on node reachability using a particular forwarding scheme and an implementation of that scheme. The purpose of this approach is to provide for an efficient development of networked systems, by accelerating the construction of their essential aspects.

To address efficiency in the modelling and analysis of network architectures, Khoury et al. [46] present a design methodology based on the Alloy language, which is based on relations and first-order predicate logic. The concept of architectural style is at the core of this formalism in order to define a precise, common design vocabulary for a class of architectures. The methodology is demonstrated by describing a model of a class of network architectures called FARA [3]. FARA is an abstract high-level network model in which the Internet architecture is modelled, to enable decoupling of end-point names from network addressing.

Another approach that deals with efficient network development is proposed by Griffin and Sobrinho [37]. They use a high-level and declarative language to model routing and its correctness properties by proposing an approach called Metarouting. The theoretical basis of Metarouting is the Routing Algebra framework of Sobrinho [62]. These formal models use a correct-by-construction approach in which the verification of convergence is accomplished once for the idealized algebra, and any routing protocol that implements the algebra is correct.

Wang et al. [69, 70] propose a formally verifiable networking (FVN) approach for unifying the design, specification, implementation and verification of networking protocols. The FVN framework uses a formal logical foundation to specify the behaviour and the properties of network protocols and of the abstract network meta-model. A theorem prover such as PVS [6] or Coq [2] is used to verify the specified formal properties of declarative network protocols.

To bridge the high-level specification with the implementation of networked systems in these related works, their authors propose new languages for which the theoretical bases are proved using formal methods. An important advantage of our method for network development consists in *reusing* an

existing language to bridge high-level specifications with implementations, namely Event-B; in our approach, we handle this bridging challenge with the powerful refinement technique.

## 5.4 Correct-by-Construction System Development

Correct-by-construction methods have been applied in several domains related to networked systems such as product line analysis, train systems, etc. Lamprecht et al. [47] use the correct-by-construction approach to propose a modelling framework for the analysis of product lines. They combine synthesis technology with a constraint-oriented approach to guarantee the validity of system properties. The validity of a variant of a product line is satisfied if properties of the high-level model of the product line are preserved. They present the applicability of their approach by illustrating it on a coffee machine example.

Moller et al. [51] verify railway systems through CSP||B modelling and analysis. In [52], they propose a structured way of analysis for interlocking railway systems by applying the correct-by-construction approach. They propose model checking of the abstract model to ensure the safety properities that hold in the concrete models as well.

# 6. Discussion

In this final section, we outline the main achievements put forward in this dissertation as well as point out future research directions.

**Summary** In this dissertation, we have aimed at contributing to the development of networked architectures for *efficiency* and *reliability*, where by efficiency we refer to the possibility of *reusing* such architectures and by reliability we refer to developing *correct* models with respect to their specifications. In order to provide reusable architectures, we need to *construct* them in certain ways; for verifying the correctness of the architectures, we need to *analyse* them with respect to certain properties. Our research method consists in applying formal methods in order to achieve our aims, in particular the *abstraction* and *refinement* techniques.

**The Challenge** A very interesting challenge in applying formal methods in the domain of networked systems consists in determining a formal theory of network architectures and, as a consequence, providing reusable, pre-proved design rules. Determining a formal theory of networked architectures requires generic models. These models consist of the essential components of any networked system together with their relations. In addition, such models need to preserve the global correctness of the network. Providing reusable, pre-proved design rules needs considering individual components of a networked system and their variations. If a correct model of a component and its variants can be developed independently of other components, then it can be reused in the development of a certain architecture.

**What we have achieved** Our goal has been to develop approaches for the formal construction and analysis of networked systems that support:

(1) A reusable and correct-by-construction development of networked systems

(2) Precise and sound foundation for rigorous qualitative and quantitative analysis

To achieve this goal, we have developed three artefacts:

(A) A formal generic architectural model for the correct development of networked systems

(B) Methods for reusable networked system development

(C) A library of rules for developing and analysing networked systems

Artefact (A), a formal generic architectural model for the correct development of networked systems, consists of high-level generic models for specifying networked systems and their properties that are preserved under refinement. General high-level models represent the conceptual framework of networked systems as a general architectural model of networks. The general architectural model can be instantiated to a particular architecture by refinement, so that the functional correctness is preserved; this is an answer to the correct development of networked systems. As network development as we proposed it does not start from scratch, but from using the general architectural model, this is also an answer to the reusable development of networked systems. Artefact (A) is therefore an answer to goal (1).

Artefact (B), consisting of the methods for reusable networked system development, defines processes in which models can be specified and shown to be reusable. The methods that are used to support reusability in this thesis are: abstraction, refinement, refinement patterns, modularisation, theory extension and automata-based templates. Artefact (B) is therefore an answer to goal (1), as well.

Artefact (C), consisting of a library of rules for developing and analysing networked systems, consists of a set of fundamental design rules which define the individual architectural components of networked systems that can be separately reused in the development and analysis of certain architectures. Artefact (C) is therefore an answer to goal (1) and (2).

**Future Directions** We can put forward several possible directions for future work. We have proposed a network framework consisting of high-level descriptions of networked systems, together with their functional properties. An interesting research direction consists in further extending of the current framework by including other features of networked systems. Such an extension could support, for instance, the analysis of real-time and non-functional properties such as power consumption, delay and performance during development. Moreover, the applicability of the framework could be examined by using it in the development and analysis of various networks. One could investigate how flexible the framework is to analyse a wider class of networked systems and to support different views of development in our framework.

These investigations would suggest the possible lines of the framework extension. In fact, by applying the framework for the development and analysis of different networked systems, two points could be highlighted: the fulfilment of the reusability feature and the clear modifications and extensions which could improve the framework usefulness.

Another research direction consists in extending the library of design rules, by adding new pre-proved network structures and techniques or refining the existing rules. Extending the library of pre-proved rules promotes the efficiency of developing reliable networked systems.

A significant challenge in our methodology for networked systems is that the qualitative and quantitative analysis are not integrated. This means that much effort is needed to model a network in such a way that either qualitative or quantitative analysis can be performed. As a solution to this, we plan to study how both qualitative and quantitative analysis of networked systems can be unified at the architectural level. Ideally, we should be able to define transformation rules that automatically transform a developed model used for qualitative analysis into a model for quantitative analysis and vice versa. Even more, the automatic transformation is conceivable at the architectural level, where the design detail are excluded. Providing an integrated tool support for developing and analysing both qualitatively and quantitatively the network architectures is also very instrumental.

# 7. Bibliography


[1] ACL2, online at http://www.cs.utexas.edu/ moore/acl2/.

[2] The coq proof assistanct, online at http://coq.inria.fr/, accessed 20.07.2013.

[3] FARA, online at http://www.isi.edu/newarch/fara.html.

[4] HOL, online at http://www.cl.cam.ac.uk/research/hvg/hol.

[5] isabelle, online at http://isabelle.in.tum.de/.

[6] PVS specification and verification system, online at http://pvs.csl.sri.com/, accessed 20.07.2013.

[7] Rodin - rigorous open development environment for complex, deliverable d7, event-b language, online at http://rodin.cs.ncl.ac.uk.

[8] SPIN, online at http://spinroot.com/spin/whatispin.html.

[9] J.-R. Abrial. *The B-book: assigning programs to meanings.* Cambridge University Press, New York, NY, USA, 1996.

[10] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[11] J.-R Abrial, M. Butler, S. Hallerstede, M. Leuschel, M. Schmalz, and L. Voisin. Proposals for mathematical extensions for event-b. Technical report, Deploy Project, 04 2010.

[12] J.-R. Abrial, M. J. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *STTT*, 12(6):447–466, 2010. `http://dx.doi.org/10.1007/s10009-010-0145-y`.

[13] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[14] M. B. Andriamiarina, H. Daoud, M. Belarbi, D. Méry, and C. Tanougast. Formal Verification of Fault Tolerant NoC-based Architecture. In *First International Workshop on Mathematics and Computer Science (IWMCS2012)*, December 2012.

[15] R. J. R. Back. Refinement calculus, part ii: Parallel and reactive programs. In *REX Workshop*, pages 67–93, 1989.

[16] R. J. R. Back and R. Kurki-Suonio. Distributed cooperation with action systems. *ACM Trans. Program. Lang. Syst.*, 10(4):513–554, October 1988.

[17] R. J. R. Back and K. Sere. From action systems to modular systems. In *Software - Concepts and Tools*, volume 17, pages 1–25. Springer-Verlag, 1994.

[18] R. J. R. Back and K. Sere. Superposition refinement of reactive systems. *Formal Aspects of Computing*, 8(3):324–346, 1996.

[19] R. J. R. Back and J. von Wright. Trace refinement of action systems. In *CONCUR*, pages 367–384, 1994.

[20] R. J. R. Back and J. V. Wright. *Refinement calculus - a systematic introduction.* Undergraduate texts in computer science. Springer, 1999.

[21] G. Behrmann, A. David, and K. G. Larsen. A tutorial on uppaal. In *SFM*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer, 2004.

[22] C. Bernardeschi, P. Masci, and H. Pfeifer. Analysis of wireless sensor network protocols in dynamic scenarios. In *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, SSS '09, pages 105–119, Berlin, Heidelberg, 2009. Springer-Verlag.

[23] K. Bhargavan, D. Obradovic, Carl, and A. Gunter. Formal verification of standards for distance vector routing protocols. *Journal of the ACM*, 49:538–576, 2002.

[24] P. Böhm. A framework for incremental modelling and verification of on-chip protocols. In *Proceedings of the 2010 Conference on Formal Methods in Computer-Aided Design*, FMCAD '10, pages 159–166, Austin, TX, 2010. FMCAD Inc.

[25] P. Böhm and T. Melham. A refinement approach to design and verification of on-chip communication protocols. In *Proceedings of the 2008*

*International Conference on Formal Methods in Computer-Aided Design*, FMCAD '08, pages 18:1–18:8, Piscataway, NJ, USA, 2008. IEEE Press.

[26] D. Borrione, A. Helmy, L. Pierre, and J. Schmaltz. A generic model for formally verifying noc communication architectures: A case study. In *Proceedings of the First International Symposium on Networks-on-Chip*, NOCS '07, pages 127–136, Washington, DC, USA, 2007. IEEE Computer Society.

[27] D. Borrione, A. Helmy, L. Pierre, and J. Schmaltz. A formal approach to the verification of networks on chip. *EURASIP J. Embedded Syst.*, 2009:2:1–2:14, January 2009.

[28] P. Bulychev, A. David, K. G. Larsen, M. Mikucionis, P. Bogsted, A. Legay, and Z. Wang. Uppaal-smc: Statistical model checking for priced timed automata. In Herbert Wiklicky and Mieke Massink, editors, *Proceedings 10th Workshop on Quantitative Aspects of Programming Languages and Systems, Tallinn, Estonia, 31 March and 1 April 2012*, volume 85 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–16. Open Publishing Association, 2012.

[29] M. Butler and I. Maamria. Mathematical extension in event-b through the rodin theory component. Technical report, Deploy Project, 10 2010.

[30] Y.-R. Chen, W.-T. Su, P.-A. Hsiung, Y.-C. Lan, Y.-H. Hu, and S.-J. Chen. Formal modeling and verification for network-on-chip. In *Green Circuits and Systems (ICGCS), 2010 International Conference on*, pages 299–304, June.

[31] A. David, K. G. Larsen, A. Legay, M. Mikucionis, and Z. Wang. Time for statistical model checking of real-time systems. In *CAV*, pages 349–355, 2011.

[32] J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors. *Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness, REX Workshop, Mook, The Netherlands, May 29 - June 2, 1989, Proceedings*, volume 430 of *Lecture Notes in Computer Science*. Springer, 1990.

[33] E. W. Dijkstra. A constructive approach to the problem of program correctness. *BIT Numerical Mathematics*, 8(3):174–186, 1968.

[34] E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1997.

[35] A. Fehnker, R.J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W.L. Tan. Automated analysis of aodv using uppaal. In *Proceedings of the 18th international conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'12, pages 173–187, Berlin, Heidelberg, 2012. Springer-Verlag.

[36] A. Fehnker, R.J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W.L. Tan. A process algebra for wireless mesh networks. In H. Seidl, editor, *European Symposium on Programming (ESOP'12)*, volume 7211 of *Lecture Notes in Computer Science*, pages 295–315. Springer, 2012.

[37] T. G. Griffin and J. L. Sobrinho. Metarouting. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '05, pages 1–12, New York, NY, USA, 2005. ACM.

[38] R. Ameur-Boulifa H. Mokrani. A refinement approach to design and verification of on-chip communication protocols. In *Proceedings of the SAFA workshop*, 2011.

[39] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, August 1978.

[40] P. Höfner and A. McIver. Statistical model checking of wireless mesh routing protocols. In *NASA Formal Methods Symposium (NFM'13)*, volume undefined of *Lecture Notes in Computer Science*, page undefined. Springer, 2013.

[41] P.-A. Hsiung and F. Wang. A state graph manipulator tool for real-time system specification and verification. In *Real-Time Computing Systems and Applications, 1998. Proceedings. Fifth International Conference on*, pages 181–188, Oct.

[42] A. Iliasov, L. Laibinis, E. Troubitsyna, and A. Romanovsky. Formal derivation of a distributed program in event b. In *Proceedings of the 13th international conference on Formal methods and software engineering*, ICFEM'11, pages 420–436, Berlin, Heidelberg, 2011. Springer-Verlag.

[43] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, and T. Latvala. Supporting reuse in event b development: modularisation approach. In *Proceedings of the Second international conference on Abstract State Machines, Alloy, B and Z*, ABZ'10, pages 174–188, Berlin, Heidelberg, 2010. Springer-Verlag.

[44] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Volume 1.* Springer Publishing Company, Incorporated, 2010.

[45] M. Karsten, S. Keshav, S. Prasad, and M. Beg. An axiomatic basis for communication. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 217–228, New York, NY, USA, 2007. ACM.

[46] J. Khoury, C. Abdallah, and G. Heileman. Towards formalizing network architectural descriptions. In Marc Frappier, Uwe Glässer, Sarfraz Khurshid, Régine Laleau, and Steve Reeves, editors, *Abstract State Machines, Alloy, B and Z*, volume 5977 of *Lecture Notes in Computer Science*, pages 132–145. Springer Berlin Heidelberg, 2010.

[47] A.-L. Lamprecht, T. Margaria, I. Schaefer, and B. Steffen. Synthesis-based variability control: Correctness by construction. In *FMCO*, pages 69–88, 2011.

[48] Y.-C Lan, S.-H. Lo, Y.-C. Lin, Y.-H. Hu, and S.-J. Chen. Binoc: A bidirectional noc architecture with dynamic self-reconfigurable channel. In *Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '09, pages 266–275, Washington, DC, USA, 2009. IEEE Computer Society.

[49] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[50] R. Milner. *Communicating and mobile systems: the &pgr;-calculus*. Cambridge University Press, New York, NY, USA, 1999.

[51] F. Moller, H. N. Nguyen, M. Roggenbach, S. Schneider, and H. Treharne. Csp||b modelling for railway verification: The double junction case study. In *AVOCS'12*, 2012.

[52] F. Moller, H. N. Nguyen, M. Roggenbach, S. Schneider, and H. Treharne. Defining and model checking abstractions of complex railway models using csp||b. In *HVC'2012*, 2012.

[53] A. Müller. Vdm — the vienna development method, 2009.

[54] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[55] V.A. Palaniveloo and A. Sowmya. Application of formal methods for system-level verification of network on chip. In *VLSI (ISVLSI), 2011 IEEE Computer Society Annual Symposium on*, pages 162–169, July.

[56] C. E. Perkins, E. M. Belding-Royer, and I. D. Chakeres. Ad hoc on-demand distance vector (AODV) routing. Internet Draft, 2003. At `http://tools.ietf.org/id/draft-perkins-manet-aodvbis-00.txt`.

[57] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *Proceedings of the conference on Communications architectures, protocols and applications*, SIGCOMM '94, pages 234–244, New York, NY, USA, 1994. ACM.

[58] J. Schmaltz and D. Borrione. A generic network on chip model. In *Proceedings of the 18th international conference on Theorem Proving in Higher Order Logics*, TPHOLs'05, pages 310–325, Berlin, Heidelberg, 2005. Springer-Verlag.

[59] S. Schneider and H. Treharne. Csp theorems for communicating b machines. *Formal Aspects of Computing*, 17(4):390–422, December 2005.

[60] S. Schneider, H. Treharne, and H. Wehrheim. A csp approach to control in event-b. In *Proceedings of the 8th international conference on Integrated formal methods*, IFM'10, pages 260–274, Berlin, Heidelberg, 2010. Springer-Verlag.

[61] K. Sen, M. Viswanathan, and G. A. Agha. Vesta: A statistical model-checker and analyzer for probabilistic systems. In *Quantitative Evaluaiton of Systems (QEST'05)*, pages 251–252. IEEE, 2005.

[62] J. L. Sobrinho. Network routing with path vector protocols: theory and applications. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 49–60, New York, NY, USA, 2003. ACM.

[63] J. M. Spivey. *The Z notation: a reference manual.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[64] T. Teige, A. Eggers, and M. Fränzle. Constraint-based analysis of concurrent probabilistic hybrid systems: An application to networked automation systems. *Nonlinear Analysis: Hybrid Systems*, 5(2):343–366, 2011.

[65] F. Verbeek and J. Schmaltz. Formal specification of networks-on-chips: deadlock and evacuation. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 1701 –1706, march 2010.

[66] F. Verbeek and J. Schmaltz. Automatic verification for deadlock in networks-on-chips with adaptive routing and wormhole switching. In *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '11, pages 25–32, New York, NY, USA, 2011. ACM.

[67] F. Verbeek and J. Schmaltz. On necessary and sufficient conditions for deadlock-free routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 22(12):2022–2032, 2011.

[68] F. Verbeek and J. Schmaltz. Easy formal specification and validation of unbounded networks-on-chips architectures. *ACM Trans. Des. Autom. Electron. Syst.*, 17(1):1:1–1:28, January 2012.

[69] A. Wang, P. Basu, B. T. Loo, and O. Sokolsky. Declarative network verification. In *Proceedings of the 11th International Symposium on Practical Aspects of Declarative Languages*, PADL '09, pages 61–75, Berlin, Heidelberg, 2009. Springer-Verlag.

[70] A. Wang, L. Jia, C. Liu, B. T. Loo, O. Sokolsky, and P. Basu. Formally verifiable networking. In Lakshminarayanan Subramanian, Will E. Leland, and Ratul Mahajan, editors, *HotNets*. ACM SIGCOMM, 2009.

[71] N. Wirth. Program development by stepwise refinement. *Communications of the ACM*, 14(4):221–227, 1971.

[72] C. Xiong, T. Murata, and J. Tsai. Modeling and simulation of routing protocol for mobile ad hoc networks using colored petri nets. In *Proceedings of the conference on Application and theory of petri nets: formal methods in software engineering and defence systems - Volume 12*, CRPIT '02, pages 145–153, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc.

[73] H. Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Carnegie Mellon University, 2004.

[74] C. Yuan, J. Billington, and J. Freiheit. An abstract model of routing in mobile ad hoc networks. In *Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 137–156, Aarhus, Denmark, October 2005. DAIMI.

# Complete List of Publications

1. Peter Höfner, and Maryam Kamali. Quantitative Analysis of AODV and its Variants on Dynamic Topologies using Statistical Model Checking, In V. Braberman and L. Fribourg (Eds.) *Proceedings of the 11th International Conference on Formal Modeling and Analysis of Timed Systems - FORMATS 13*, Lecture Notes in Computer Science Vol. 8053, pp. 121-136, Springer-Verlag, 2013.

2. Ansgar Fehnker, Peter Höfner, Maryam Kamali, and Vinay Mehta. Topology-based Mobility Model for Wireless Networks, In K. Joshi et al. (Eds.) *Proceedings of the 10th International Conference on Quantitative Evaluation of Systems conference - QEST 13*, Lecture Notes in Computer Science Vol. 8054, pp. 389-404, Springer-Verlag, 2013.

3. Maryam Kamali, Mats Neovius, Luigia Petre and Petter Sandvik. Formal Development of System of Systems, Accepted to *ISRN Software Engineering Journal*, Hindawi Publishing Corporation, 2013. (In press)

4. Maryam Kamali, Linas Laibinis, Luigia Petre, and Kaisa Sere. A Distributed Implementation of a Network Recovery Algorithm, In *International Journal of Critical Computer-Based Systems (IJCCBS)*, Vol. 4, No. 1, pp. 45-68. Inderscience Publishers, 2013.

5. Maryam Kamali, Luigia Petre, and Kaisa Sere. NetCorre: A Hierarchical Framework and Theory for Network Design (*Submitted to Science of Computer Programming Journal*), April, 2013.

6. Maryam Kamali, Linas Laibinis, Luigia Petre, and Kaisa Sere. Formal Development of Wireless Sensor-Actor Networks, In *Science of Computer Programming (SCP) Journal.* Elsevier, 2012. DOI: 10.1016/j.scico.2012.03.002.

7. Maryam Kamali, Luigia Petre, Kaisa Sere, and Masoud Daneshtalab. Refinement-Based Modeling of 3D NoCs, In F. Arbab and M. Sirjani (Eds.) *Proceedings of the 4th IPM International Conference on*

*Fundamentals of Software Engineering - FSEN 11*, Lecture Notes in Computer Science Vol. 7141, pp. 236-252, Springer-Verlag, 2012.

8. Masoud Daneshtalab, Maryam Kamali, Masoumeh Ebrahimi, Saeed Mohammadi, Ali Afzali-Kusha, Juha Plosila. Adaptive Input-Output Selection Based On-Chip Router Architecture, In *Journal of Low Power Electronics*, Volume 8, Number 1, pp. 11-29, February 2012.

9. Maryam Kamali, Luigia Petre, Kaisa Sere, and Masoud Daneshtalab. CorreComm: A Formal Hierarchical Framework for Communication Designs, In *Proceedings of the 2nd IEEE International Conference on Networked Embedded Systems for Enterprise Applications - NESEA2011*, pp. 1-7. IEEE Computer Society, December, 2011.

10. Maryam Kamali, Luigia Petre, Kaisa Sere, and Masoud Daneshtalab. Formal Modeling of Multicast Communication in 3D NoCs. In P. Kitsos and S. Niar (Eds.) *Proceedings of the 14th Euromicro Conference on Digital System Design - DSD 2011*, pp. 634-642. IEEE/Euromicro, August 2011.

11. Maryam Kamali, Luigia Petre, Kaisa Sere, Masoud Daneshtalab. A Formalization of 3D NoCs, In M. Waldén, and L. Petre (Eds.) *Proceedings of the 22nd Nordic Workshop on Programming Theory - NWPT2010*, TUCS Technical Reports, No. 57, pp. 79-81, November 2010.

12. Maryam Kamali, Linas Laibinis, Luigia Petre and Kaisa Sere. Self-Recovering Sensor-Actor Networks, In M. Mousavi and G. Salaün (Eds.) *Proceedings of the 9th International Conference on the Foundations of Coordination Languages and Software Architectures - FOCLASA2010*, pp. 47-61. EPTCS, September, 2010.

13. Maryam Kamali, Linas Laibinis, Luigia Petre and Kaisa Sere. Reconstructing Coordination Links in Sensor-Actor Networks, In C. Jensen (Eds.) *Proceedings of the 4th Nordic Workshop on Dependability and Security - NODES 2010*, pp. 1-10, June 2010.

14. Mohsen Sharifi, Saeed Sedighian and Maryam Kamali. Recharging Sensor Nodes Using Implicit Actor Coordination in Wireless Sensor Actor Networks, In *Scientific Research Journal of Wireless Sensor Network*, Vol 2, N. 2, pp. 123-131, February 2010.

15. Maryam Kamali, Mohsen Sharifi and Saeed Sedighian. A Distributed Recovery Mechanism for Actor-Actor Connectivity in Wireless Sensor Actor Networks, In *Proceedings of the 4nd IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing - ISSNIP 2008*, pp. 183-188, December 2008.

# Part II

# Original Publications

# Paper I

## Formal Development of Wireless Sensor-Actor Networks

**Maryam Kamali, Linas Laibinis, Luigia Petre and Kaisa Sere**

Originally published in: *Science of Computer Programming (SCP) Journal*, Elsevier, 2012. In press, DOI: 10.1016/j.scico.2012.03.002.

Based on the publication: Maryam Kamali, Linas Laibinis, Luigia Petre and Kaisa Sere. Self-Recovering Sensor-Actor Networks, In *Proceedings of the 9th International Conference on the Foundations of Coordination Languages and Software Architectures - FOCLASA2010*, pp. 47-61. EPTCS, September, 2010.

ARTICLE IN PRESS
Science of Computer Programming ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

# Formal development of wireless sensor–actor networks

Maryam Kamali [a,b,∗], Linas Laibinis [a], Luigia Petre [a], Kaisa Sere [a]

[a] Department of Information Technologies, Åbo Akademi University, Turku, Finland
[b] Turku Centre for Computer Science (TUCS), Turku, Finland

ARTICLE INFO

*Article history:*
Received 22 January 2011
Received in revised form 30 January 2012
Accepted 5 March 2012
Available online xxxx

*Keywords:*
Wireless sensor–actor networks (WSANs)
Coordination links
Coordination recovery
Refinement
Pattern development
Event-B
RODIN tool

ABSTRACT

Wireless sensor–actor networks are a recent development of wireless networks where both ordinary sensor nodes and more sophisticated and powerful nodes, called *actors*, are present. In this paper we introduce several, increasingly more detailed, formal models for this type of wireless networks. These models formalise a recently introduced algorithm for recovering actor–actor coordination links via the existing sensor infrastructure. We prove via refinement that this recovery is correct and that it terminates in a finite number of steps. In addition, we propose a generalisation of our formal development strategy, which can be reused in the context of a wider class of networks. We elaborate our models within the Event-B formalism, while our proofs are carried out using the RODIN platform — an integrated development framework for Event-B.

© 2012 Elsevier B.V. All rights reserved.

## 1. Introduction

The separation of computation and control stands at the basis of the software architecture discipline. The control of the *computing* entities as well as the self-coordination of the *controlling* entities are well illustrated by Wireless Sensor–Actor Networks (WSANs), a rather new generation of sensor networks [1]. WSAN nodes can be of two types: *sensors* (the 'computing' entities) and *actors* (the controlling entities), with the density of sensor nodes much bigger than that of actor nodes. The sensors detect the events that occur in the field, gather them and transmit the collected data to the actors. The actors react to the events in the environment based on the received information. The sensor nodes are low-cost, low-power devices equipped with limited communication capabilities, while the actor nodes are usually mobile, more sophisticated and powerful devices compared to the sensor nodes.

A central WSAN requirement is that of node *coordination*. As there is no centralised control in a WSAN, sensors and actors need to coordinate with each other in order to collect information and take decisions on the following actions [1]. There are three main types of WSAN coordination [2]: sensor–sensor, sensor–actor and actor–actor coordinations. The sensor–sensor coordination in WSANs is similar to that of Wireless Sensor Networks, i.e., it defines how sensors route information, how information aggregates among them and which sensors are responsible for which tasks. The sensor–actor coordination prescribes which sensors should send certain data to which actors. Finally, the actor–actor coordination is concerned with actor decisions and the division of tasks among different actors. In this paper, we focus on the latter type of coordination.

To achieve the actor–actor coordination in WSANs, actors need reliable connection links for communicating with each other. These are established upon initialising a WSAN. However, WSANs are dynamic networks where the network topology continuously changes. The changes occur when new links or nodes are added or when the existing links or nodes are removed

∗ Corresponding author at: Department of Information Technologies, Åbo Akademi University, Turku, Finland.
*E-mail address:* mkamali@abo.fi (M. Kamali).

0167-6423/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2012.03.002

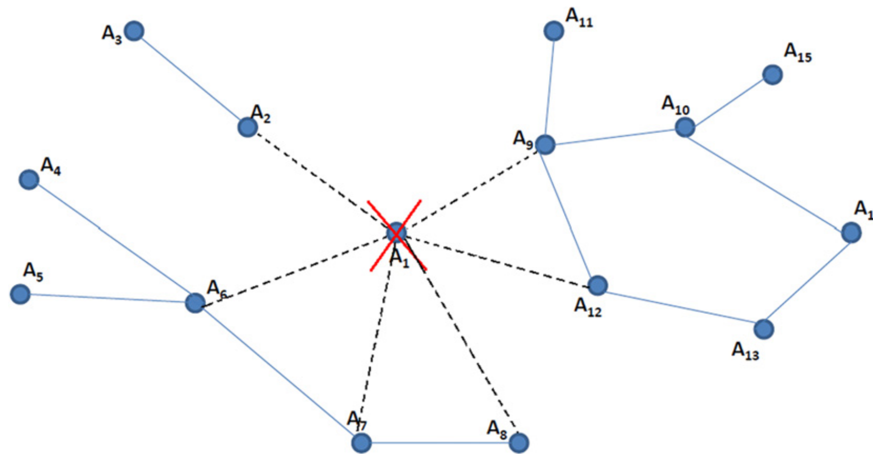



**Fig. 1.** Three partitions created by the failed actor $A_1$.

due to failures, typically caused by hardware crashes or malfunctions, the lack of energy, mobility, etc. Consequently, actor nodes can fail during operation of a network. As a result, a WSAN may transform into several, disconnected WSAN sub-networks, thus interrupting the actor–actor coordination. Such a separation is called a *network partitioning* and is illustrated in Fig. 1, where the actor nodes $A_1$–$A_{15}$ are shown to produce a network partitioning if the actor node $A_1$ fails.

Another central WSAN requirement focuses on embedding *real-time aspects*. In particular, depending on an application, it might be essential to respond to sensor inputs within predefined time limits, e.g., in critical applications such as forest fire detection. Due to the real-time requirements of WSANs, a failure of an actor node should not affect the whole actor network for too long. To re-establish connectivity of actor nodes, their physical movement towards each other has been proposed in [3,4]. However, during this movement, nodes in different network partitions created by an actor failure cannot coordinate. To shorten the time of recovery, Kamali et al. [5] have proposed an algorithm for establishing new routes between non-failed actors via sensor nodes. This algorithm allows for quick reconnection of the separated partitions, *before* moving actor nodes as proposed in [3,4]. In this paper we focus on studying this recovery mechanism, which alleviates the problems caused by actor coordination failures.

There are several properties that are desirable to verify for this algorithm. First, we need to show that there is *always* a path via sensor nodes that *can be* established for the partitioned actor nodes. Second, it is important to guarantee that the new path is the shortest one for the involved actor nodes. Assuming that the sensor network is sufficiently dense, this also reduces the power consumption of the sensor nodes that are employed to re-establish connection. Third, to shorten the time of recovery as much as possible, it is desirable to re-establish connection as soon as possible. In this paper we address the first and the second property of the algorithm.

The contribution of our paper is threefold. First, we formalise the algorithm for self-recovering actor coordination [5] using a theorem prover tool. This allows us to better understand the functioning of the algorithm and, more importantly, to verify essential properties such as the functional correctness and termination of the recovery mechanism. An important aspect of the recovery is that indirect links between actors are built in a *distributed* manner, thus ensuring *self* -recovering of the network. Second, we prove that the recovery can be done at different levels, via different link types, such as direct or indirect actor links, in the latter case also reusing the WSAN infrastructure of sensors. This contributes to modelling fault-recovery in sensor–actor networks. Third, we explore a generalisation of the described approach to a wider class of networks by using the notions of refinement patterns and pattern-driven formal development. This allows us to facilitate the presented formal development process by identifying the development (design) steps typical for coordinating networks as well as reusing both formal models and proofs.

To model the functional correctness of the recovery algorithm, we use the mathematical concepts of *tree* and *forest*. In graph theory, a tree is a graph whose any two vertices are connected by a non-cyclic path, while a forest is a set of disjoint trees. As special cases, an empty graph (with no nodes) and a discrete graph on a set of vertices (with no edges) are examples of forests. We introduce a special data structure to model a forest and use it to prove correctness properties in the following way. When a node fails, the set of all the neighbours of the failed node is considered as a set of disconnected trees, i.e., a (node) forest. For instance, in Fig. 1 the nodes $A_2$, $A_6$, $A_7$, $A_8$ $A_9$, and $A_{12}$ are the neighbours of the failed node $A_1$. These nodes are considered as disconnected trees, with each tree initially formed of exactly one of the nodes. While the recovery process either finds or re-establishes links, these trees gradually become connected to each other. When the recovery process terminates, we show that all the trees of the forest are connected. This means that, by the end of the recovery, all the neighbours of the failed node are re-connected.

In order to prove correctness of the recovery mechanism, we employ the Event-B formalism for modelling WSANs and the proposed algorithm [5]. Event-B [6,7] is an extension of the B Method [8] for specifying distributed and reactive systems. In Event-B, a system model is gradually specified at several levels of abstraction, always ensuring that a more concrete model is a *correct development* of the previous, more abstract model. The language and proof theory of Event-B are based on predicate logic and the set theory. Correctness of stepwise construction of formal models is ensured by discharging a

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

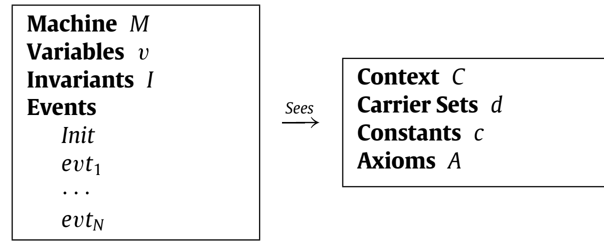



**Fig. 2.** A machine $M$ and a context $C$ in Event-B.

set of proof obligations: if these obligations hold, then the undertaken development is mathematically shown to be correct. Event-B comes with the associated tool RODIN [7,9,10,18], which automatically discharges a part of the proof obligations and also provides the interactive prover to discharge the remaining proofs.

The paper is organised as follows. In Section 2 we discuss the Event-B formalism and also describe the recovery algorithm. In Section 3 we overview our formal development of the recovery mechanism. Sections 4–7 contain the description of our formal models at four abstraction levels. Specifically, in Section 4 we model the direct actor link recovery mechanism. In Section 5 the abstract model is enhanced by the indirect actor link recovery. In Section 6 we introduce the sensor infrastructure and the indirect actor link recovery via the introduced sensors. In Section 7 we model physical distances between nodes to ensure the shortest indirect links via sensors. We continue in Section 8 by discussing proving aspects regarding our formal development. In Section 9 we bring forward our approach to generalisation and reuse of the introduced formal models. We survey related work in Section 10 and in Section 11 we present several concluding remarks.

## 2. Preliminaries

This section presents the background material for the paper. We start by overviewing the Event-B formalism. In the second part of the section, we describe the recovery algorithm for WSANs, which will be formalised later.

### 2.1. Introduction to Event-B

The B Method [8] is a formal approach for the specification and rigorous development of highly dependable software. The method has been successfully used in the development of several complex real-life applications [10,11]. Event-B [6] is a formal framework derived from the B Method to model and reason about parallel, distributed and reactive systems. Event-B has the associated RODIN platform [7,9,10,18], which provides automated tool support for modelling and verification by theorem proving.

**Event-B Language.** In Event-B, a system specification (model) is defined using the notion of a *machine* [10] operating on an *abstract state*. Such a machine encapsulates the model state, represented as a collection of model variables, and defines operations on this state. Thus, it describes the behaviour of the modelled system, also referred to as the dynamic part. A machine may also have an accompanying component, called *context*, which contains the static part of the system. A context can include user-defined carrier sets, constants and their properties, which are given as a list of model axioms. The general form of an Event-B model is illustrated in Fig. 2. The relationship between a machine and its accompanying context is expressed by the keyword *Sees*, denoting a structuring technique that allows the machine access to the contents of the context.

A machine is uniquely identified by its name $M$. The state variables, $v$, are declared in the **Variables** clause and initialised by the *Init* event. The variables are strongly typed by the constraining predicates $I$ given in the **Invariants** clause. The invariant clause may also contain other predicates defining properties that should be preserved over the state of the model.

The dynamic behaviour of the system is defined by a set of atomic events specified in the **Events** clause. Generally, an event can be defined as follows:

$$evt \ \widehat{=}\ \textbf{any}\ vl\ \textbf{where}\ g\ \textbf{then}\ S\ \textbf{end},$$

where the variable list $vl$ contains new local variables (parameters) of the event, the guard $g$ is a conjunction of predicates over the state variables $v$ and $vl$, and the action $S$ is an assignment to the state variables.

The occurrence of events represents the observable behaviour of the system. The event guard defines the conditions under which the action can be executed, i.e., when the event is *enabled*. If several events are enabled at the same time, any of them can be chosen for execution non-deterministically. If none of the events is enabled then the system deadlocks.

In general, the action of an event is a parallel composition of assignments. The assignments can be either deterministic or non-deterministic. A deterministic assignment, $x := E(x, y)$, has the standard syntax and meaning. A non-deterministic assignment is denoted either as $x :\in Set$, where $Set$ is a set of values, or $x :| \ P(x, y, x')$, where $P$ is a predicate relating initial values of $x, y$ to some final value $x'$. As a result of a non-deterministic assignment, $x$ can get any value belonging to $Set$ or according to $P$.

| Action $(S)$ | $BA(S)$ |
|---|---|
| $x := E(x, y)$ | $x' = E(x, y) \wedge y' = y$ |
| $x :\in Set$ | $x' \in Set \wedge y' = y$ |
| $x :\mid P(x, y, x')$ | $P(x, y, x') \wedge y' = y$ |

**Fig. 3.** Before–after predicates.

**Event-B Semantics.** The semantics of Event-B actions is defined using so-called *before–after (BA) predicates* [6,10]. A before–after predicate describes a relationship between the system states before and after execution of an event, as shown in Fig. 3. Here $x$ and $y$ are disjoint lists (partitions) of state variables, and $x'$, $y'$ represent their values in the after-state. A before–after predicate for events is constructed as follows:

$$BA(evt) = \exists vl.\, g \wedge BA(S).$$

The semantics of a whole Event-B model is formulated as a number of *proof obligations*, expressed in the form of logical sequents. Below we describe only the most important proof obligations that should be verified (proved) for the initial and refined models. The full list of proof obligations can be found in [6].

Every Event-B model should satisfy the event consistency and invariant preservation properties. For each event of the model, $evt_i$, its consistency means that, whenever the event is enabled, its before–after predicate (BA) is well-defined, i.e., there exists some reachable after-state:

$$A(d, c),\ I(d, c, v),\ g_i(d, c, v) \vdash \exists v' \cdot BA_i(d, c, v, v') \quad \text{(FIS)}$$

where $A$ stands for the conjunction of the model axioms, $I$ is the conjunction of the model invariants, $g_i$ is the event guard, $d$ stands for the model sets, $c$ are the model constants, and $v$, $v'$ are the variable values before and after the event execution.

Each event $evt_i$ of the Event-B model should also preserve the given model invariant:

$$A(d, c),\ I(d, c, v),\ g_i(d, c, v),\ BA_i(d, c, v, v') \vdash I(d, c, v') \quad \text{(INV)}$$

Since the initialisation event has no initial state and guard, its proof obligation is simpler:

$$A(d, c),\ BA_{Init}(d, c, v') \vdash I(d, c, v') \quad \text{(INIT)}$$

The formal semantics provides us with a foundation for establishing correctness of Event-B specifications. In particular, to verify correctness of a specification, we need to prove that its initialisation and all the events preserve the given invariant.

**System development.** Event-B employs a top-down refinement-based approach to formal system development. Development starts from an abstract system specification that models some of the essential functional requirements. While capturing more detailed requirements, each refinement step typically introduces new events and variables into an abstract specification. These new events correspond to stuttering steps that are not visible in the abstract specification. We call such model refinement as *superposition refinement*. Moreover, Event-B formal development supports data refinement, allowing us to replace some abstract variables with their concrete counterparts. In that case, the invariant of a refined model formally defines the relationship between the abstract and concrete variables; this type of invariant is called a *gluing invariant*. Out of these refinement approaches, in this paper we employ the superposition refinement.

When presenting events in a refined model, we often use the shorthand notation "*refined_event* **extends** *abstract_event*". The meaning of this notation is that the refined event is created from the abstract one by simply adding new guards and/or new actions. Only the added elements are shown in the extended event, while the old guards and actions are implicitly present.

To verify correctness of a refinement step, we need to prove a number of proof obligations for a refined model. For brevity, here we show only a few essential ones.

Let us first introduce a shorthand $H(d, c, v, w)$ to stand for the hypotheses $A(d, c),\ I(d, c, v),\ I'(d, c, v, w)$, where $I,\ I'$ are respectively the abstract and refined invariants, and $v$, $w$ are respectively the abstract and concrete variables. Then the consistency refinement property for an event $evt_i$ of a refined model can be presented as follows:

$$H(d, c, v, w),\ g'_i(d, c, w) \vdash \exists w'.\, BA'_i(d, c, w, w') \quad \text{(REF_FIS)}$$

where $g'_i$ is the refined guard and $BA'_i$ is a before–after predicate of the refined event.

The event guards in a refined model can be only strengthened in a refinement step:

$$H(d, c, v, w),\ g'_i(d, c, w) \vdash g_i(d, c, v) \quad \text{(REF_GRD)}$$

where $g_i$, $g'_i$ are respectively the abstract and concrete guards of the event $evt_i$.

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

ARTICLE IN PRESS


```
0 // Detecting a failed actor; Failed_Actors and Active_Actors partition the set Actors
1  IF actor i, i ∈ Active_Actors receives no acknowledgement from actor j,
2       j ∈ Neighbours(i), for a period t
3  THEN Failed_Actors := Failed_Actors ∪ {j}
4       Active_Actors := Active_Actors \ {j}
5  FI
6  FailedNodeNeighbours := Neighbours(j)
7  IF ∀ k,l ∈ FailedNodeNeighbours
8       there is a path from k to l via actors distinct from j
9  THEN Neighbours(k) := Neighbours(k) \ {j}
10      Neighbours(l) := Neighbours(l) \ {j}
11      FailedNodeNeighbours := FailedNodeNeighbours \ {k}
12 ELSE recovery(FailedNodeNeighbours)
13 FI
14 // Selecting the shortest distance between neighbours and establishing it via sensors
15 PROCEDURE recovery(valres FailedNodeNeighbours)
16 FOR ∀ k ∈ FailedNodeNeighbours with
17     degree(k) = max{degree(i)|i ∈ FailedNodeNeighbours}
18   find an actor l ∈ FailedNodeNeighbours so that
19     dist(k,l) = min{dist(k,i)|i ∈ FailedNodeNeighbours \ {k}}
20   establish the new communication link between k and l through sensors
21   FailedNodeNeighbours := FailedNodeNeighbours \ {k}
22 END FOR
23 END PROCEDURE
```

**Fig. 4.** A general pseudo-code of the recovery algorithm.

Finally, the *simulation* proof obligation requires to show that the "execution" of a refined event is not contradictory to its abstract version:

$$H(d, c, v, w),\ g_i'(d, c, w),\ BA_i'(d, c, w, w') \vdash \exists v'.BA_i(d, c, v, v') \wedge I'(d, c, v', w') \qquad \text{(REF_SIM)}$$

where $BA_i$, $BA_i'$ are respectively the abstract and concrete before–after predicates of the same event $evt_i$.

The Event-B refinement process allows us to gradually introduce implementation details, while preserving functional correctness during stepwise model transformation. The model verification effort and, in particular, automatic generation and proving of the required proof obligations, are significantly facilitated by the provided tool support — the RODIN platform [7,9,10,18].

Let us note here the quintessential feature of Event-B and its associated RODIN platform. Modelling in Event-B is semantically justified by proof obligations. Every update of a model generates a new set of proof obligations in the background. It is this interplay between modelling and proving that sets Event-B apart from other formalisms. Without proving the required obligations, we cannot be sure of correctness of a model. The proving effort thus encourages the developer to structure formal model development in such a way that manageable proof obligations are generated at each step. This leads to very abstract initial models so that we can gradually introduce into a system model various facets of the system. Such a development method fits well when we have to describe complex algorithms.

### 2.2. The recovery mechanism

Our object of study is the recovery algorithm introduced in [5]. A rather general pseudo-code model of the algorithm is shown in Fig. 4, while a more detailed model appears in [5].

In this algorithm, the detection of a failed actor leads to the communication links among non-failed actor nodes to be re-established via sensor nodes. The mechanism has three parts: detecting a failed actor (lines 1–13), selecting the shortest path via actors (lines 15–19), and establishing the selected path through sensor nodes (line 20).

Actors are modelled in the pseudo-code with the set `Actors`, which is further partitioned into the sets `Active_Actors` and `Failed_Actors`. When actor neighbours of an actor node do not receive any acknowledgement from that actor node, they detect it as failed and move it from the `Active_Actors` set to the `Failed_Actors` set (lines 1–5). All the neighbours of the failed node are collected in the set `FailedNodeNeighbors` (line 6). At this time, the neighbours of the failed node have to investigate this set to determine whether the actor failure has produced separate partitions. No partitioning means there is a path between any two actors in `FailedNodeNeighbors`, via the actors distinct from the failed actor.

If there is no partitioning (lines 7–8), then nothing is done except updating the neighbour lists of nodes (lines 9–11). However, if separate partitions are detected, a new path is selected and established, which is modelled by the procedure `recovery` (lines 15–23). This procedure takes (as a value-result parameter) the set `FailedNodeNeighbors`, which is first analysed (lines 16–19) and then updated (line 21).

In this algorithm, modelling the actor–actor coordination is based on the assumption that each actor node has information about its immediate neighbours (1-hop neighbours) and 2-hop neighbours (the neighbours of the neighbours) [5]. This is the least costly coordination model with respect to energy consumption, because it involves the least number of nodes required to re-establish communication. Based on this information, the non-failed actors can recover their communication links upon detecting a failed (intermediary) actor. These links are formed based on the node `degree` information (the number of

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

immediate neighbours) and on the relative distance `dist` between actor nodes. We formally detail this in Sections 4–7, while in the next section we overview the overall formal development.

### 3. Skeleton of the development

Our initial goal in this paper is to prove that the recovery mechanism described in the previous section works. For this, we have created a formal model of the algorithm in Event-B and proved that the algorithm indeed reconnects partitions in a finite number of steps after an actor failure. The model is created in four increasingly more detailed refinement steps. In this section we overview the design decisions that we made during the refinement process as well as describe the main data structures and variables of our models. While developing the models, we have found that our development of the recovery mechanism in a WSAN is more general than we initially thought. We explore this generality in more detail in Section 9.

The wireless sensor–actor networks that we model employ distributed recovery that is location-based and hierarchical [12]. In order to better focus on the task at hand, we exclude from our model several WSAN features such as energy-saving mechanisms and mobility of nodes. Our assumption for the recovery to succeed is that there are sufficiently densely deployed sensors in the field. Another assumption is that only one actor can fail at one time. The recovery mechanism takes place almost immediately and the recovery process is done in a negligible time, hence this assumption does not indeed restrict our model. Moreover, we note that we do not model time to effectively measure duration of the recovery, nor do we simulate our algorithm in this paper. Modelling time and thus demonstrating the third desirable property of the algorithm as described in Section 1 is left for future research. A simulation of the original algorithm can be found in [5].

We model a wireless sensor–actor network as a graph $(NODE, Net)$, where $NODE$ is a set of nodes modelling both sensors and actors and $Net$ is a set consisting of all direct communication links between nodes. We assume that communication is symmetric, meaning that if a node can hear another node, then it can also be heard by it. The corresponding graph is undirected. A wireless link exists between any two nodes only if these nodes are within wireless range of each other:

$$Net = \{(u, v) \in NODE \times NODE \mid dist(u, v) \leq R\}$$

Here, $R$ is the communication range among any two nodes and $dist(u, v)$ denotes the Euclidean distance between the nodes $u$ and $v$. Namely, if $u_x$ and $u_y$ are the horizontal and vertical Cartesian coordinates of the node $u$, respectively, and $v_x$ and $v_y$ are the horizontal and vertical Cartesian coordinates of the node $v$, respectively, then $dist(u, v) = \sqrt{(u_x - v_x)^2 + (u_y - v_y)^2}$. The $Net$ relation models the links between all the nodes in $NODE$, without considering their sensor/actor type.

In our modelled recovery mechanism, we distinguish among three types of network link: sensor–sensor, sensor–actor, and actor–actor. To model them, we partition the network nodes into two different types of node, i.e., sensors and actors: $NODE = sensors \cup actors$. Based on this partition, we then also partition the direct network links into three types:

$$Net = SNet \cup ANet \cup SANet,$$

where $SNet$ denotes direct links between sensors, $ANet$ denotes direct links between actors, and $SANet$ denotes direct links between sensors and actors. The network graph $(NODE, Net)$ is therefore partitioned as follows:

$$(NODE, Net) = (sensors, SNet) \cup (actors, ANet) \cup (sensors \cup actors, SANet)$$

$$SNet = \{(u, v) \in sensors \times sensors \mid dist(u, v) \leq R_s\}$$

$$ANet = \{(u, v) \in actors \times actors \mid dist(u, v) \leq R_a\}$$

$$SANet = \{(u, v) \in (sensors \cup actors) \times (sensors \cup actors) \mid dist(u, v) \leq R_s\}$$

Here, $R_a$ is the actor communication range and $R_s$ is the sensor communication range. Here we assume that $R_s \leq R_a$ and require that every direct link between a sensor and an actor to respect the range $R_s$. We also assume that $R_s \leq R_a \leq R$.

In order to prove the correctness of the algorithm, we do not model it in all detail from the beginning. Instead, we start with a rather abstract model, which is still sufficient to prove various important properties for the algorithm. Then we gradually add the required details to that initial model. Such an abstraction strategy proves to be essential to our development.

The main purpose of the recovery algorithm is to re-establish connections among the partitions that are formed by an actor failure. Initially, we specify an abstract recovery model that re-establishes new connection routes after an actor failure. In fact, the abstract model does not include a specific recovery algorithm. The initial model is kept general, as a model of fault tolerance that has the potential to be refined to a particular recovery mechanism. We observe that, by keeping the initial model very abstract, we can derive a pattern from it and reuse this pattern for modelling and verifying other models of fault tolerance.

Due to these reasons, in our first model we only describe the actor network, without any knowledge about sensor nodes and their links. With this partial knowledge about the network, we model the recovery mechanism non-deterministically and in a centralised manner. Namely, we assume that global knowledge about 1-hop and 2-hop links (direct and indirect) between actors is available. This global knowledge is modelled by the relation $ANet$ for the 1-hop links and by the relation $ANet ; ANet$ for the 2-hop links (here $ANet ; ANet$ models the forward composition of relation $ANet$ with itself). Furthermore, addition of new actor links in the initial model is also kept abstract, considering neither their communication range nor their location coordinates.

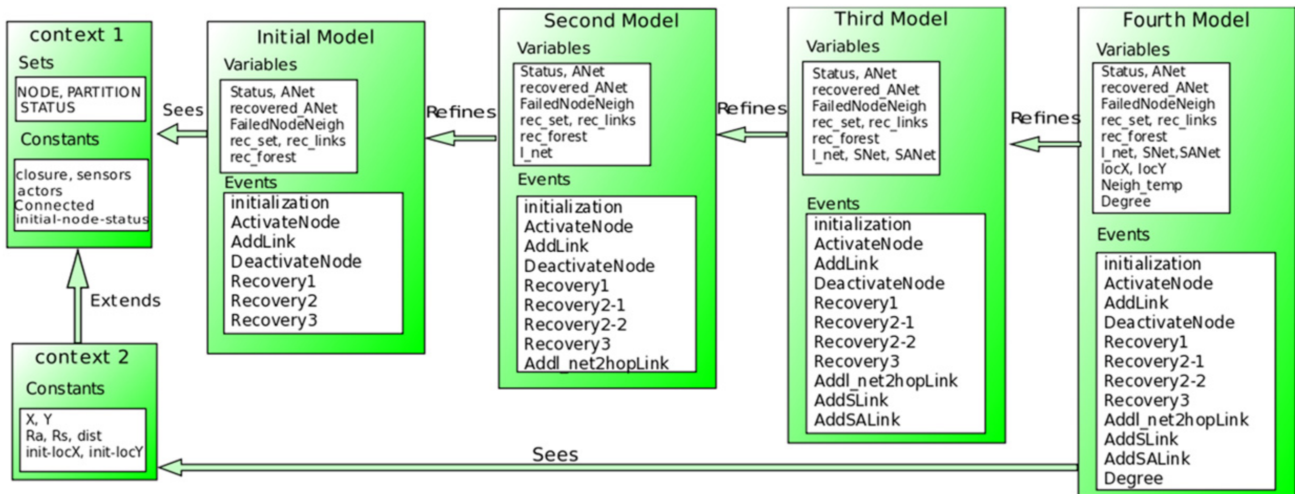



**Fig. 5.** Overview of model development.

In the second model we focus on the distributed nature of the recovery mechanism. For this, we restrict the global knowledge about the network topology to the information about 1-hop and 2-hop neighbours. Specifically, the recovery mechanism is based on the knowledge stored in a variable *l_net* of the type $(actors \times actors) \leftrightarrow NODE$. Assuming that $(n, m), (m, k) \in ANet$, then a tuple $(n, m, m) \in l_net$ stores the direct, 1-hop link $(n, m)$ and a tuple $(n, k, m) \in l_net$ stores the indirect, 2-hop link $(n, m); (m, k)$. We prove that our recovery mechanism works distributively, i.e., it is based only on the localised knowledge stored in the *l_net* variable.

In the third model, we add the earlier described *SNet* and *SANet* relations that model all the links between sensors and between sensors and actors. Moreover, the recovery mechanism is refined to model the re-establishing of actor links through sensors.

Finally, in the fourth model, the actor and sensor communication ranges are introduced and the location coordinates of nodes are specified. Links among nodes are established only if the nodes are within the communication ranges. In addition, we employ the Euclidean distance for selecting the shortest path among two actors that are neighbours of a failed node. Once this path is found, the recovery between the actors is performed as described in the third step, i.e., via sensors.

Let us observe that our fourth model contains all the details of the recovery algorithm described in the previous section. However, the order of introducing these details is different from the order in which the algorithm proceeds in Section 2.2. For instance, the algorithm is composed of three sequential parts, such as detecting a failed actor, selecting the shortest path via actors, and establishing the selected path through sensor nodes. We detect a failed actor in the first model and then use the detection throughout the other three models, while we establish the path through sensors in the third model, and finally select the shortest path via actors in the fourth model.

This particular order of introducing details into a model is chosen because it produces logical and manageable proof obligations. In addition, each refinement step is a small increment of the previous step, as this tends to help us manage the proofs better with the tool rather when many details are introduced in a step. For instance, we could have introduced the location coordinates already in the first model and selected the shortest path already in the same model. However, the prover would then have generated many obligations that were not necessary at that stage. Similarly, adding wireless links according to communication ranges can occur earlier in the model development, but we chose to introduce it when all the network links were modelled, in order to simplify and reuse the proofs as much as possible.

The graphical overview of our gradual model construction is shown in Fig. 5. The boxes contain data structure names, variable names, and event names. We will describe all these entities in detail in the following sections.

## 4. The initial model: recovery via direct actor links

We start a description of our formal development with the initial, abstract network model representing essential network functionality as well as a simple recovery mechanism via direct actor links. First we introduce necessary concepts in the model context component.

### *4.1. Model context*

The context of our initial model contains definitions of constants and sets as well as our model assumptions as axioms on the introduced sets and constants. We note here that the RODIN tool does not include support for verifying the consistency of the model axioms. It is therefore a responsibility of the modeller to ensure this.

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

ARTICLE IN PRESS



The set of all the network nodes is modelled as a finite (**@axm1**), non-empty (**@axm2**), generic set *NODE*. These nodes can be either sensor nodes or actor nodes, hence we partition the set *NODE* into two subsets, *sensors* and *actors*, as described by **@axm3-6**. Also we introduce an abstract set *PARTITION* to distinguish between partitions in the network.

We also define another generic set $STATUS = \{ok, fail\}$, where the constant *fail* models the failed status of a node and the constant *ok* models the active status of a node (**@axm7**). In addition, we introduce a constant *closure* that models the transitive closure of a binary relation on the set *NODE* (**@axm8-11**).

**CONSTANTS** *closure connected ok fail sensors actors initial_node_status*
**SETS** *NODE PARTITION STATUS*
**AXIOMS**
**@axm1** $finite(NODE)$
**@axm2** $NODE \neq \varnothing$
**@axm3** $sensors \subseteq NODE$
**@axm4** $actors \subseteq NODE$
**@axm5** $sensors \cap actors = \varnothing$
**@axm6** $sensors \cup actors = NODE$
**@axm7** $partition(STATUS, \{ok\}, \{fail\})$
**@axm8** $closure \in (NODE \leftrightarrow NODE) \rightarrow (NODE \leftrightarrow NODE)$
**@axm9** $\forall r \cdot r \subseteq closure(r)$
**@axm10** $\forall r \cdot closure(r); r \subseteq closure(r)$
**@axm11** $\forall r, s \cdot r \subseteq s \wedge s; r \subseteq s \Rightarrow closure(r) \subseteq s$

The constant *closure* is employed to dynamically construct all the indirect sensor links in the current network. Moreover, it allows us to express the property of network nodes being connected. Specifically, a function (predicate) *connected* is defined (**@axm12-13**) for this purpose. The function takes a set of nodes and a set of network links and returns a Boolean value indicating whether all the given nodes are connected via the given links. The main definition of *connected* (**@axm13**) relies on the existence of a link path between arbitrary nodes $n$ and $m$ from the given set, i.e., the pair $n \mapsto m$ belongs to the transitive closure on the given links.

The essential properties of *connected* are listed as the axioms **@axm14-18**. The consistency of these properties has been proven in the interactive theorem prover HOL [19]. The most important property for proving the correctness of the proposed recovery mechanism is **@axm14**, which states that two disconnected networks become connected by adding links between two arbitrary nodes belonging to these separate networks. We note that the other axioms of our model (**@axm1-13**) either denote standard mathematical properties or constraints of our model.

Finally, the initial status of the network nodes is given as an abstract constant *initial_node_status* (**@axm19**). It can be considered as a parameter of our formal development.

**AXIOMS**
**@axm12** $connected \in \mathbb{P}(NODE) \times (NODE \leftrightarrow NODE) \rightarrow BOOL$
**@axm13** $\forall S, L \cdot (connected(S \mapsto L) = TRUE) \Leftrightarrow (\forall n, m \cdot n \in S \wedge m \in S \Rightarrow (n \mapsto m \in closure(L)))$
**@axm14** $\forall S1, S2, L1, L2, n, m \cdot n \in S1 \wedge m \in S2 \wedge connected(S1 \mapsto L1) = TRUE$
$\wedge\ connected(S2 \mapsto L2) = TRUE \Rightarrow connected((S1 \cup S2) \mapsto (L1 \cup L2 \cup \{n \mapsto m, m \mapsto n\})) = TRUE$
**@axm15** $\forall L \cdot connected(\varnothing \mapsto L) = TRUE$
**@axm16** $\forall L, n \cdot n \in dom(L) \Rightarrow connected(\{n\} \mapsto L) = TRUE$
**@axm17** $\forall S, L1, L2 \cdot L1 \subseteq L2 \wedge connected(S \mapsto L1) = TRUE \Rightarrow connected(S \mapsto L2) = TRUE$
**@axm18** $\forall S1, S2, L \cdot S1 \subseteq S2 \wedge connected(S2 \mapsto L) = TRUE \Rightarrow connected(S1 \mapsto L) = TRUE$
**@axm19** $initial_node_status \in NODE \rightarrow STATUS$

## 4.2. Model variables and invariants

In the machine part of our initial model we introduce seven variables as follows. Their invariant properties are listed below (in three separate boxes).

**VARIABLES**
*Status, ANet, recovered_ANet, rec_set, FailedNodeNeigh, rec_links, rec_forest*

**INVARIANTS**
**@inv1** $Status \in NODE \rightarrow STATUS$
**@inv2** $ANet \in actors \leftrightarrow actors$
**@inv3** $\forall n, m \cdot n \mapsto m \in ANet \Rightarrow Status(n) = ok \wedge Status(m) = ok$
**@inv4** $actors \lhd id \cap ANet = \varnothing$
**@inv5** $ANet = ANet^{-1}$
**@inv6** $recovered_ANet \in actors \leftrightarrow actors$
**@inv7** $\forall n, m \cdot n \mapsto m \in recovered_ANet \Rightarrow Status(n) = ok \wedge Status(m) = ok$
**@inv8** $actors \lhd id \cap recovered_ANet = \varnothing$
**@inv9** $recovered_ANet = recovered_ANet^{-1}$
**@inv10** $ANet \cap recovered_ANet = \varnothing$
**@inv11** $FailedNodeNeigh \subseteq actors$
**@inv12** $FailedNodeNeigh \subseteq (Status^{-1}[\{ok\}])$

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

The dynamic status of each node (active or failed) is modelled with a function *Status*, which maps each node in *NODE* to *ok* or *fail* (**@inv1**). A relation *ANet* denotes bidirectional, non-failed actor links (**@inv2** and **@inv3**). This relation is required to be irreflexive (**@inv4**) and symmetric (**@inv5**). This means that an *ANet* link from an actor to itself is prohibited and, if an actor *a* has a link with an actor *b*, the actor *b* also has a link with the actor *a*. Let us note that the relation *ANet* stores only direct links between actors.

For our development purposes, we define a relation *recovered_ANet* to store the indirect links that are established by the recovery mechanism after a failure of an actor node. As *ANet*, *recovered_ANet* is irreflexive and symmetric. Moreover, *recovered_ANet* links only active nodes, and *ANet* and *recovered_ANet* are disjoint (**@inv6-10**).

The recovery process is triggered by a detection of a failed actor node. The active nodes directly affected by a failure are stored in a set variable *FailedNodeNeigh*. Specifically, *FailedNodeNeigh* contains active actor neighbours of a failed actor (**@inv11-12**). The set is repeatedly updated during the recovery process, as also illustrated by the pseudo-code in Fig. 4.

...
**@inv13** $rec_set \subseteq actors$
**@inv14** $rec_set \subseteq (Status^{-1}[\{ok\}])$
**@inv15** $FailedNodeNeigh \subseteq rec_set$
**@inv16** $rec_links \in rec_set \leftrightarrow rec_set$
**@inv17** $rec_links = rec_links^{-1}$
**@inv18** $\forall n, m \cdot n \mapsto m \in rec_links \wedge n \neq m \Rightarrow n \mapsto m \in (ANet \cup recovered_ANet \cup (ANet;ANet))$
**@inv19** $rec_forest \in PARTITION \leftrightarrow rec_set$
**@inv20** $rec_forest^{-1} \in rec_set \rightarrow PARTITION$

To model and verify the recovery algorithm, which re-establishes connection among the neighbours of a failed node after its failure, we define three extra variables *rec_set*, *rec_links* and *rec_forest*. These variables will be later directly involved in defining desired properties of the recovery algorithm as model invariants.

The subset *rec_set* models all the active neighbours of a failed actor (**@inv13-15**). This set is equal to *FailedNodeNeigh* when the recovery starts. However, unlike *FailedNodeNeigh*, it remains constant throughout the recovery process. The relation *rec_links* denotes all the links between the neighbours of the failed node that can be established via *ANet*, *ANet*; *ANet* or *recovered_ANet* (**@inv16-18**). Finally, the relation *rec_forest* models the separated network partitions caused by a node failure as a node forest. The variable *rec_forest* is repeatedly updated during the recovery process when nodes gradually discover alternative routes and thus reduce the network partitioning (**@inv19-20**).

...
**@inv21** $\forall p1 \cdot p1 \in dom(rec_forest) \Rightarrow connected(rec_forest[\{p1\}] \mapsto rec_links) = TRUE$
**@inv22** $FailedNodeNeigh \neq \varnothing \Rightarrow card(dom(rec_forest)) = card(FailedNodeNeigh)$
**@inv23** $card(FailedNodeNeigh) = 0 \wedge card(rec_set) \neq 0 \Rightarrow card(dom(rec_forest)) = 1$

The last three invariants (**@inv21-23**) express the desired correctness properties of the recovery algorithm. The invariant **@inv21** states that, for each element of a node forest modelled as the relation *rec_forest*, all its nodes are connected to each other by the existing links in *rec_links*. In other words, *rec_forest* indeed represents a forest consisting of node trees. This property is preserved during the recovery process when new links are added and, as a result, the forest elements are merged together.

The set *FailedNodeNeigh* models the active neighbours of a failed node, which have not yet discovered or re-established a link between each other. In each step of the recovery process, a node from *FailedNodeNeigh* is selected, the links associated with this node and other disconnected neighbours are re-established, and, finally, the node is removed from *FailedNodeNeigh*. As a result, each recovery step decreases the number of forest elements by one. In fact, during the recovery process the number of forest elements is always equal to that of *FailedNodeNeigh*, as formulated in the invariant **@inv22**.

When the recovery terminates, *FailedNodeNeigh* becomes empty. The invariant (**@inv23**) states that this happens only when *rec_forest* represents a single node tree. In other words, all the nodes in *rec_set* have now a route to each other via *rec_links*. This essentially gives us a proof of correctness of the modelled recovery algorithm since it relates the end of the recovery with the reconnection of all the actor nodes whose neighbour has previously failed.

### 4.3. Model events

The initial model contains six events (in addition to the obligatory initialisation event).

**INITIALISATION**
**BEGIN**
**@act1** $Status := initial_node_status$
**@act2** $ANet := \varnothing$
**@act3** $recovered_ANet := \varnothing$
**@act4** $rec_links := \varnothing$
**@act5** $rec_forest := \varnothing$
**@act6** $FailedNodeNeigh := \varnothing$
**@act7** $rec_set := \varnothing$
**END**

**ActivateNode**
**ANY** n **WHERE**
**@grd1** $n \in actors$
**@grd2** $Status(n) = fail$
**@grd3** $FailedNodeNeigh = \varnothing$
**THEN**
**@act1** $Status(n) := ok$
**END**

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

ARTICLE IN PRESS


The **INITIALISATION** event sets the status of all the nodes based on the given constant *initial_node_status* and, as a result, builds the initial configuration of the network. The relations *ANet*, *recovered_ANet*, *rec_links* and *rec_forest* are all initialised to be empty. The sets *FailedNodeNeigh* and *rec_set* are initialised to empty sets as well.

The other events in the initial model activate actor nodes from failed to active (**ActivateNode**), add actor links (**AddLink**), deactivate actor nodes and remove their corresponding links (**DeactivateNode**), and abstractly recover connections when an actor fails (**Recovery1**, **Recovery2** and **Recovery3**). Thus, starting from the initial setting (given as *initial_node_status*) consisting of failed and active actors that have no links between each other, we can randomly activate actors, add links between active actors, as well as deactivate actors and remove their corresponding links.

The **DeactivateNode** event models actor failures and thus enables our recovery mechanism. Until the recovery is complete, the first three events (**ActivateNode**, **AddLink**, and **DeactivateNode**) are not enabled anymore. Consequently, we have the normal operation phase of the network, when the **ActivateNode**, **AddLink**, and **DeactivateNode** events are non-deterministically executed, and the recovery phase of the network, when only **Recovery1**, **Recovery2** and **Recovery3** events are executed. The phase separation is modelled using the variable *FailedNodeNeigh*. While *FailedNodeNeigh* is empty, the network is in its operational phase. Otherwise, the network is in its recovery phase.

In the **ActivateNode** event shown above, the status of a failed actor (see the event guards **@grd1** and **@grd2**) is changed to active (the action **@act1**). The event is enabled if *FailedNodeNeigh* is empty (**@grd3**), thus this is an event of the operational phase.

**AddLink**
**ANY** n m **WHERE**
**@grd1** $n \in actors \wedge m \in actors$
**@grd2** $Status(n) = ok \wedge Status(m) = ok$
**@grd3** $n \neq m$
**@grd4** $n \mapsto m \notin ANet$
**@grd5** $FailedNodeNeigh = \varnothing$
**THEN**
**@act1** $ANet := ANet \cup \{n \mapsto m, m \mapsto n\}$
**@act2** $recovered_ANet := recovered_ANet \setminus \{n \mapsto m, m \mapsto n\}$
**END**

In the **AddLink** event, we add a link between two distinct, active actors (**@grd1-3**) that are not connected (**@grd4**). To respect the invariants **@inv5** and **@inv10**, we also add the links in both directions in *ANet* (**@act1**) and remove them from the relation *recovered_ANet* (**@act2**). This corresponds to cancelling the temporary links proposed by the recovery algorithm. When two nodes can be connected directly, an indirect link consuming more power is not needed. Therefore, the constructed indirect links are removed from *recovered_ANet*. The event is enabled only if *FailedNodeNeigh* is empty (**@grd5**), thus this is again an event of the operational phase. In later refinement models we restrict adding communication links based on the communication range of the actors.

**DeactivateNode**
**ANY** n i **WHERE**
**@grd1** $n \in actors$
**@grd2** $Status(n) = ok$
**@grd3** $FailedNodeNeigh = \varnothing$
**@grd4** $i \in ANet[\{n\}] \rightarrowtail PARTITION$
**@grd5** $i^{-1} \in PARTITION \twoheadrightarrow ANet[\{n\}]$
**THEN**
**@act1** $Status(n) := fail$
**@act2** $ANet := \{n\} \lhd\!\!\!- ANet \,-\!\!\!\rhd \{n\}$
**@act3** $recovered_ANet := \{n\} \lhd\!\!\!- recovered_ANet \,-\!\!\!\rhd \{n\}$
**@act4** $FailedNodeNeigh := ANet[\{n\}]$
**@act5** $rec_forest := i^{-1}$
**@act6** $rec_set := ANet[\{n\}]$
**@act7** $rec_links := (ANet[\{n\}] \lhd id)$
**END**

The event **DeactivateNode** changes the status of an active actor (**@grd1-2**) to that of a failed one (**@act1**). Also, all the links of the actor are removed from both *ANet* and *recovered_ANet*, which is expressed by using correspondingly the domain subtraction operator $\lhd\!\!\!-$ and the range subtraction operator $-\!\!\!\rhd$ in the event actions **@act2-3**. By removing these links, we preserve the invariant property **@inv3**. In addition, the neighbours of the actor are included into the set *FailedNodeNeigh* (**@act4**).

In order to verify correctness of the recovery algorithm, we construct a node forest, all elements of which consists of individual nodes (the neighbours of the failed node), as required by **@grd4-5**. As a result, the number of the failed node's neighbours is the number of forest elements at the start of the recovery process. The constructed forest becomes a new value of the relation *rec_forest* (**@act5**). At the same time, the variable *rec_set* is assigned the set consisting of all the failed node's neighbours (**@act6**). Finally, the variable *rec_links* is re-initialised for recovery by linking all the neighbour nodes to themselves (**@act7**). Consequently, the event **DeactivateNode** acts as an initialisation event for the recovery process.

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

**DeactivateNode** is an event of the operational phase, enabled only when *FailedNodeNeigh* is empty (**@grd3**). If the failed actor has neighbours in *ANet*, *FailedNodeNeigh* becomes not empty after **@act4** is executed. At this point, the network enters into the recovery phase. As mentioned in Section 3, we only model the situations where one actor fails at a time, therefore **DeactivateNode** cannot become enabled again until the recovery for this failed node is finished.

**Recovery1**
**ANY** n k **WHERE**
  **@grd1** $n \in FailedNodeNeigh \wedge k \in FailedNodeNeigh \wedge n \neq k$
  **@grd2** $n \mapsto k \notin ANet \wedge n \mapsto k \notin ANet; ANet$
**THEN**
  **@act1** $recovered_ANet := recovered_ANet \cup \{n \mapsto k, k \mapsto n\}$
  **@act2** $FailedNodeNeigh := FailedNodeNeigh \setminus \{n\}$
  **@act3** $rec_links := rec_links \cup \{n \mapsto k, k \mapsto n\}$
  **@act4** $rec_forest := (\{rec_forest^{-1}(n)\} \lhd\!\!\!- rec_forest)$
    $\cup(\{rec_forest^{-1}(k)\} \times rec_forest[\{rec_forest^{-1}(n)\}])$
**END**

Deactivating an actor triggers the recovery process. The event **Recovery1** is enabled when two neighbours of the failed actor (**@grd1**) have no connection through other neighbours (**@grd2**) (there is no path from one actor to another considering at most 2-hop distance). We note that this check does not imply that deactivation of an actor necessarily leads to partitioning of the actor network, although in some cases it may.

A full check for the actor network partitioning would require the guard **@grd2** to be of the form $n \mapsto k \notin closure(ANet)$. However, this is a very strong condition for an actor node, since no actor can have such global knowledge on the entire actor network. Hence, we enable our recovery phase if there are no short (maximum 2-hop) paths among the neighbours of the deactivated actor.

When the event **Recovery1** is executed, a direct actor–actor link is established and stored separately in the relation *recovered_ANet* (**@act1**). This separation is essential because *ANet* models only the direct links between nodes. The recovered links modelled by *recovered_ANet* are 'magically' re-established direct links. In other words, the linked nodes are not within the communication range of each other and thus they need, in fact, to communicate through some intermediate nodes, which are not yet modelled at this level of abstraction.

When a link is established between two nodes belonging to the set *FailedNodeNeigh*, this link is also added to the relation *rec_links* (**@act3**). Moreover, the two separated elements of the node forest, denoted by *n* and *k* are now merged to a joint one (**@act4**), so that their corresponding node trees are now connected through the added links. At the same time, a neighbour node is removed from the set *FailedNodeNeigh* (**@act2**).

**Recovery2**
**ANY** n k **WHERE**
  **@grd1** $n \in FailedNodeNeigh \wedge k \in FailedNodeNeigh$
  **@grd2** $n \mapsto k \in ANet \vee n \mapsto k \in (ANet; ANet) \setminus (actors \lhd id)$
**THEN**
  **@act1** $FailedNodeNeigh := FailedNodeNeigh \setminus \{n\}$
  **@act2** $rec_links := rec_links \cup \{n \mapsto k, k \mapsto n\}$
  **@act3** $rec_forest := (\{rec_forest^{-1}(n)\} \lhd\!\!\!- rec_forest)$
    $\cup(\{rec_forest^{-1}(k)\} \times rec_forest[\{rec_forest^{-1}(n)\}])$
**END**

The event **Recovery2** deals with the situation when a failure is detected but an alternative path through 1-hop or 2-hop neighbours already exists between the neighbours of the deactivated actor, i.e.,

$$n \mapsto k \in ANet \vee n \mapsto k \in ANet; ANet \setminus (actors \lhd id).$$

In this case, *FailedNodeNeigh* is simply updated by removing the corresponding node. Adding links to *rec_links* and merging elements of a node forest in *rec_forest* are similar to **Recovery1**.

**Recovery3**
**WHERE**
  **@grd1** $card(FailedNodeNeigh) = 1$
**THEN**
  **@act1** $FailedNodeNeigh := \varnothing$
**END**

The event **Recovery3** concludes the recovery process by removing the last remaining node from *FailedNodeNeigh*. According to the invariant **@inv23**, all the forest elements are now merged into a single node tree.

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

ARTICLE IN PRESS


**Table 1**
Mapping between elements of the original algorithm and the initial model.

| Elements of algorithm | Elements of initial model |
|---|---|
| `FailedNodeNeighbours` | *FailedNodeNeigh* |
| `Actors` | *actors* |
| `Failed_Actors` | $Status^{-1}[\{fail\}]$ |
| `Active_Actors` | $Status^{-1}[\{ok\}]$ |
| lines 1–2 | $FailedNodeNeigh \neq \varnothing$ |
| lines 3–4 | **@act1** in the event **DeactivateNode** |
| line 6 | **@act4** in the event **DeactivateNode** |
| lines 7–8 | **@grd1-2** in the event **Recovery2** |
| lines 9–10 | **@act2** in the event **DeactivateNode** |
| line 11 | **@act1** in the event **Recovery2** |
| line 12 | **@grd1-2** in the event **Recovery1** |
| line 21 | **@act2** in the event **Recovery1** |

### 4.4. Model properties

**theorem @THM1**$(\exists n \cdot\ n \in actors\ \wedge\ Status(n) = fail \wedge\ FailedNodeNeigh = \varnothing)$
$\vee\ (\exists\, n, m \cdot n \in actors\ \wedge\ m \in actors\ \wedge Status(n) = ok\ \wedge\ Status(m) = ok \wedge\ n \neq m$
$\quad \wedge\, n \mapsto\ m \notin ANet\ \ \wedge FailedNodeNeigh = \varnothing)$
$\vee\ (\exists\, n \cdot n \in actors\ \wedge\ Status(n) = ok\ \wedge\ FailedNodeNeigh = \varnothing)$
$\vee\ (\exists\, n, k \cdot n \in FailedNodeNeigh\ \wedge\ k \in FailedNodeNeigh \wedge n \neq k \wedge n \mapsto\ k \notin ANet$
$\quad \wedge n \mapsto\ k \notin (ANet; ANet))$
$\vee\ (\exists\, n, k \cdot n \in FailedNodeNeigh\ \wedge\ k \in FailedNodeNeigh \wedge\ (n \mapsto\ k \in ANet$
$\quad \vee\, n \mapsto\ k \in (ANet; ANet) \setminus (actors \lhd id)))$
$\vee\ (card(FailedNodeNeigh) = 1)$

Besides the correctness property discussed above, we show that the overall model presented in this section is deadlock-free, i.e., at any moment at least one event is enabled. We state this property as theorem **@THM1**, which is proved once for the whole model (unlike model invariants proven separately for each event). The theorem simply states that the disjunction of the guards of all the events is always true. In fact, we prove this type of theorem once for each refined model, i.e., also for the models in Sections 5–7.

Let us note that each recovery event removes a node from the set *FailedNodeNeigh*. Since *FailedNodeNeigh* is a finite set (**@axm1**, **@axm4**, **@inv11**), the recovery events can be enabled only a finite number of times. This means that the recovery phase, triggered by a node failure, always terminates. Technically, we show this by proving that the natural number expression $card(FailedNodeNeigh)$ is decreased by each execution of **Recovery1**, **Recovery2**, and **Recovery3**.

### 4.5. The original algorithm and our initial model

The mapping between elements of the algorithm pseudo-code, given in Fig. 4, and the initial model is presented in Table 1. There are several elements from Fig. 4 that are missing in our initial model. The reason for that is that the presented model is still quite abstract, yet containing all the necessary elements to enable us to later introduce various facets of the original algorithm. Thus, lines 16–20 from the pseudo-code have no correspondence in the initial model, but will have correspondence in the subsequent models.

Overall, the initial model presented in this section describes non-deterministic activation and deactivation of actor nodes as well as adding and removing of actor links in a dynamic (wireless sensor–actor) network. In such a network, communication problems between the actor nodes are detected and recovered from via 'magically' re-established direct actor links. The recovery process assumes some global network knowledge, specifically, accessing data for calculating $ANet; ANet$. Recovery finishes by non-deterministically establishing direct links among the active actor neighbours of the failed actor. Such a recovery mechanism can be used in practice only for *strategic actors*, i.e., the actor nodes whose range is sufficiently large to check the contents of *ANet* and $ANet; ANet$ and re-establish direct actor links. The following models consider more localised assumptions as well as indirect recovery paths.

Let us observe that, after the initialisation of our actor network, we have an arbitrary distribution of failed and active nodes. The active nodes can be later connected via arbitrary links using the **AddLink** event. We do not aim our recovery at the initially failed nodes, since there is nothing to recover there: those nodes have no previous links that no longer function due to some node failure. Instead, our recovery is concerned only with deactivation (failure) of actor nodes that occurs during functioning of the network.

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

## 5. First refinement: recovery via indirect actor links

In the previous section we have presented a formal model of the actor network recovery based on some global knowledge about the network links. In this section we show how we can refine the presented abstract model so that each actor has access only to information of its 1-hop neighbours and 2-hop neighbours, i.e, by modelling the actor–actor coordination recovery using local information.

The goal of this refinement step is to supplement the global knowledge about the network used in the initial model with the localised information for every actor. We achieve this by introducing a new variable, the relation $l_net$, that for each actor keeps track of the 1-hop and 2-hop neighbours and the recovered links between them that are stored in the abstract model by $recovered_ANet$.

**@inv24** $l_net \in (actors \times actors) \leftrightarrow NODE$
**@inv25** $actors \lhd id \cap dom(l_net) = \varnothing$
**@inv26** $\forall n, m . n \mapsto m \in ANet \leftrightarrow n \mapsto m \mapsto m \in l_net$
**@inv27** $\forall n, m, k . n \neq m \land m \neq k \land n \mapsto k \mapsto m \in l_net \Rightarrow$
$(n \mapsto k \in recovered_ANet \cup ((recovered_ANet \cup ANet); (recovered_ANet \cup ANet)))$
**@inv28** $recovered_ANet \subseteq dom(l_net)$

The relation $l_net$ is irreflexive (**@inv25**) and it relates a pair of actor nodes via some other (not necessarily actor) node (**@inv24**). The meaning of this relation is that any 1-hop neighbour $m$ of an actor node $n$ is denoted by $n \mapsto m \mapsto m \in l_net$, while a 2-hop neighbour $m$ of an actor $n$ is denoted by $n \mapsto m \mapsto k \in l_net$. In the first case, $m$ is locally related to $n$ via itself (by a direct link). In the second case, $m$ is locally related to $n$ via $k$ ($m$ is a 2-hop neighbour of $n$, while $k$ is either a 1-hop neighbour of $n$ or an intermediate node discovered by the recovery mechanism). In such a way, the relation $l_net$ describes all the *localised* links between nodes.

All links in $ANet$ form 1-hop links of $l_net$: **@inv26** is thus a gluing invariant. The connection between 2-hop links in $l_net$ and the global structures $ANet$ and $recovered_ANet$ is more intricate and expressed by the gluing invariant **@inv27**. As a result, we use $l_net$ in all the places where global information about the network is needed.

**AddLink**
**extends** AddLink
**THEN**
**@act3** $l_net := (l_net \setminus ((\{n\} \times \{m\} \times Status^{-1}[\{ok\}] \setminus ANet[\{n\}]) \cup$
$(\{m\} \times \{n\} \times Status^{-1}[\{ok\}] \setminus ANet[\{m\}]))) \cup \{n \mapsto m \mapsto m, m \mapsto n \mapsto n\}$
**END**

When a new link is added between two actors, the relation $l_net$ also needs to be updated. Therefore, the event **AddLink** is extended to add links to $l_net$. For every two actors $n$ and $m$ that have a direct link, $n \mapsto m \mapsto m$ and $m \mapsto n \mapsto n$ are added into $l_net$ (**@act3**), which means that $m$ is a 1-hop neighbour of $n$ and vice versa. If there exist any recovered links, they are removed from the $l_net$ relation. The expression (used in the action **@act3**)

$$\{m\} \times \{n\} \times Status^{-1}[\{ok\}] \setminus ANet[\{m\}]$$

results in all the links between $m$ and $n$ that are not 1-hop or 2-hop neighbours of $m$, i.e., they belong to the recovered links in $l_net$.

**Addl_net2hopLink**
**ANY** n m k **WHERE**
**@grd1** $Status(n) = ok \land Status(m) = ok \land Status(k) = ok$
**@grd2** $m \mapsto k \mapsto k \in l_net \land m \mapsto n \mapsto n \in l_net \land$
$n \mapsto k \mapsto m \notin l_net \land k \mapsto n \mapsto m \notin l_net$
**@grd3** $m \neq n \land n \neq k$
**@grd4** $FailedNodeNeigh = \varnothing$
**THEN**
**@act1** $l_net := l_net \setminus ((\{k\} \times \{n\} \times (Status^{-1}[\{ok\}] \setminus ANet[\{k\}])) \cup$
$(\{n\} \times \{k\} \times (Status^{-1}[\{ok\}] \setminus ANet[\{n\}]))) \cup \{n \mapsto k \mapsto m, k \mapsto n \mapsto m\}$
**END**

The event **Addl_net2hopLink** is a newly introduced event that handles the addition of 2-hop neighbour links for actor nodes. If an actor has a direct link with two other actors, these actors become 2-hop neighbours of each other. The actors involved in this event have to be active (**@grd1**) and distinct (**@grd3**). Moreover, the pairs $(m, k)$ and $(m, n)$ have a direct (1-hop) link, but a 2-hop link between $n$ and $k$ does not yet belong to $l_net$ (**@grd2**). As a result of the event, if there are any 2-hop links between $n$ and $m$ re-established through the recovery, these links are removed from $l_net$, while the newly detected 2-hop links are added to $l_net$ (**@act1**).

**DeactivateNode**
**extends** DeactivateNode
**THEN**
**@act8** $l_net := l_net \setminus ((\{n\} \times dom(ANet) \times Status^{-1}[\{ok\}])$
$\cup (dom(ANet) \times \{n\} \times Status^{-1}[\{ok\}] \setminus ANet[\{n\}]))$
**END**

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

When deactivating an actor node, all its links are also removed from $l_net$. Therefore, the event **DeactivateNode** is extended by a new action **@act8** removing the links to and from the failed actor from the relation $l_net$. The expression (used in the added action)

$$\{n\} \times dom(ANet) \times Status^{-1}[\{ok\}]$$

describes all the links from $n$, either direct (1-hop neighbours) or indirect (2-hop neighbours), as well as the recovered links. The expression

$$dom(ANet) \times \{n\} \times Status^{-1}[\{ok\}] \setminus ANet[\{n\}]$$

describes all the links to $n$ that are either direct connections (1-hop neighbours) or recovered ones.

In the following we model detection of failed actors and recovery of node links based on the introduced local information instead of the global actor–actor coordination data described by $ANet; ANet$. As a result, the $l_net$ information is used in addition to $ANet$ for detecting an actor failure (**@grd3**) and recovering links in the event **Recovery1** presented below.

**Recovery1**
**extends** Recovery1
**ANY** n m k **WHERE**
  **@grd3** $n \mapsto k \mapsto m \in l_net \wedge n \mapsto m \mapsto m \notin l_net \wedge k \mapsto n \mapsto m \in l_net \wedge k \mapsto m \mapsto m \notin l_net$
  **@grd4** $n \mapsto k \mapsto k \notin l_net \wedge k \mapsto n \mapsto n \notin l_net$
**THEN**
  **@act5** $l_net :| \; l_net' \subseteq (l_net \setminus \{n \mapsto k \mapsto m, k \mapsto n \mapsto m\}) \cup$
  $(\{n\} \times \{k\} \times Status^{-1}[\{ok\}] \setminus ANet[\{n\}]) \cup (\{k\} \times \{n\} \times Status^{-1}[\{ok\}] \setminus ANet[\{k\}]) \cup$
  $indir(k, n) \cup indir\tilde{}(n, k) \cup indir(n, k) \cup indir\tilde{}(k, n)$
**END**

When an actor $m$ is detected as failed, the neighbours of $m$ (say, $n$ and $k$) that are connected to each other via $m$ ($n \mapsto k \mapsto m$ and $k \mapsto n \mapsto m$ belong to $l_net$) need to find an alternative route for communication. If there is no other route in $ANet$ ($n \mapsto k \notin ANet \wedge n \mapsto k \notin ANet; ANet$) (as modelled by **@grd2** of **Recovery1** in the abstract model), $l_net$ should be updated by both removing invalid links and adding new ones. Since the node $m$ is marked as failed, the links between $n$ and $k$ via $m$ are not valid anymore. As a result, both $n \mapsto k \mapsto m$ and $k \mapsto n \mapsto m$ are removed from $l_net$.

The relation $l_net$ should be also updated by adding new links to connect $n$ and $k$. In the refined model, we define that an actor $n$ can establish a link with an actor $k$ through any active node which is not a neighbour of $n$, i.e., the new link belongs to $\{n\} \times \{k\} \times Status^{-1}[\{ok\}] \setminus ANet[\{n\}]$. Similarly, for the actor $k$ to establish a new link with the actor $n$, this new link should belong to $\{k\} \times \{n\} \times Status^{-1}[\{ok\}] \setminus ANet[\{k\}]$.

In addition, we have to update $l_net$ with 2-hop links involving the neighbours of $n$ and $k$. For the sake of illustration, we will use the following short definitions throughout the paper:

$$indir(k, n) = ANet[\{k\}] \times \{n\} \times \{k\}$$

and

$$indir\tilde{}(n, k) = \{n\} \times ANet[\{k\}] \times \{k\}$$

Here $indir(k, n)$ and $indir\tilde{}(n, k)$ describe the 2-hop links involving the neighbours of $k$.

When the node $n$ establishes a link with $k$, the neighbours of $n$ also need to add node $k$ to their 2-hop neighbours list (expressed as $indir(n, k)$ and $indir\tilde{}(k, n)$ respectively). Moreover, the neighbours of $k$ need to add $n$ to their 2-hop neighbours list (expressed as $indir(k, n)$ and $indir\tilde{}(n, k)$ respectively). The whole required update of $l_net$ is described by **@act5** in the event **Recovery1**.

**Recovery2-1**
**extends** Recovery2
**ANY** n m k **WHERE**
  **@grd3** $(n \mapsto k \in dom(l_net \setminus \{n \mapsto k \mapsto m\}) \wedge n \mapsto m \mapsto m \notin l_net$
  $\wedge\, k \mapsto n \in dom(l_net \setminus \{k \mapsto n \mapsto m\}) \wedge k \mapsto m \mapsto m \notin l_net)$
  $\vee (n \mapsto k \mapsto k \in l_net \wedge k \mapsto n \mapsto n \in l_net)$
**THEN**
  **@act4** $l_net := l_net \setminus (\{n \mapsto k \mapsto m, k \mapsto n \mapsto m\} \cup indir(n, m) \cup indir(k, m))$
**END**

**Recovery2-2**
**extends** Recovery2
**ANY** n m k **WHERE**
  **@grd3** $n \mapsto k \notin dom(l_net \setminus \{n \mapsto k \mapsto m\}) \wedge n \mapsto m \mapsto m \notin l_net$
  $\wedge\, k \mapsto n \notin dom(l_net \setminus \{k \mapsto n \mapsto m\}) \wedge k \mapsto m \mapsto m \notin l_net$
  $\wedge\, n \mapsto k \mapsto k \notin l_net \wedge k \mapsto n \mapsto n \notin l_net$
**THEN**
  **@act4** $l_net :| \; l_net' \subseteq (l_net \setminus \{n \mapsto k \mapsto m, k \mapsto n \mapsto m\})$
  $\cup (\{k\} \times \{n\} \times Status^{-1}[\{ok\}] \setminus ANet[\{n\}]) \cup (\{n\} \times \{k\} \times Status^{-1}[\{ok\}] \setminus ANet[\{n\}])$
  $\cup\, indir(k, n) \cup indir\tilde{}(n, k) \cup indir(n, k) \cup indir\tilde{}(k, n)$
**END**

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

The event **Recovery2** is refined by two events to consider two different cases involving *l_net* when there is a link between two neighbours either through *ANet* or *ANet*; *ANet*. The first case describes the situation when there is a 2-hop link between two *FailedNodeNeigh* nodes which is already included in *l_net*, while the second case covers the situation when there is a 2-hop link between them which is not yet included in *l_net*. In the former case, we update *l_net* by removing all the links with the failed actor or via it. In the latter case, we remove invalid links and add new routes into *l_net*. Refining **Recovery2** in such a way allows us to re-establish a new link via *l_net* without using *recovered_ANet*.

The relation *l_net* is an elegant data structure relating two actor nodes in its domain via a third node in its range. The model described in this section is a refinement of the (more abstract) model presented in the previous section. This means that the old invariants still hold for the extended model, in addition to the five new ones. Moreover, the link recovery mechanism terminates in a finite number of steps. Indirect links between actors are now established non-deterministically based on the localised information. These types of links can be further refined to a more deterministic form, as we show in the next section.

The pseudo-code in Fig. 4 has no counterpart in the model in this section, because we have not shown any information about 1-hop and 2-hop neighbours in Fig. 4. These details appear only in a more detailed version of the pseudo-code in [5].

The refined model is a specific distributed actor–actor coordination model using the *l_net* data structure. The actor–actor coordination recovery mechanism is based on this local coordination model. It is also shown that when 2-hop node neighbours, i.e., indirect links in *l_net*, are not updated, additional new links can be established. However, adding more recovered links through intermediate nodes requires extra power consumption in the whole network. Since power consumption is one of the main constraints in WSANs, this can be considered as a weakness of the recovery algorithm.

## 6. Second refinement: sensor-based recovery

In the previous model, we have defined *l_net* as a special relational structure for actor nodes. After detection of an actor failure, *l_net* had to be updated non-deterministically, due to the lack of knowledge about sensor nodes. In this refined model, we add sensor nodes and specify concretely how replacement links through sensors are added after detecting an actor failure. For this, we introduce two new relations on *NODE*, *SNet* (**@inv29**) and *SANet* (**@inv30**), the former representing the links between sensor nodes and the latter modelling the links between sensor and actor nodes.

**INVARIANTS**
**@inv29** $SNet \in sensors \leftrightarrow sensors$
**@inv30** $SANet \in NODE \leftrightarrow NODE$
**@inv31** $SNet \cap ANet = \varnothing$
**@inv32** $ANet \cap SANet = \varnothing$
**@inv33** $SNet \cap SANet = \varnothing$
**@inv34** $SNet = SNet^{-1}$
**@inv35** $SANet = SANet^{-1}$
**@inv36** $sensors \lhd id \cap SNet = \varnothing$
**@inv37** $NODE \lhd id \cap SANet = \varnothing$
**@inv38** $\forall n, m \cdot n \mapsto m \in SANet \Rightarrow$
$(n \in actors \land m \in sensors) \lor (m \in actors \land n \in sensors)$
**@inv39** $\forall n, m \cdot n \mapsto m \in SNet \Rightarrow Status(n) = ok \land Status(m) = ok$
**@inv40** $\forall n, m \cdot n \mapsto m \in SANet \Rightarrow Status(n) = ok \land Status(m) = ok$
**@inv41** $\forall n, k, x, y \cdot n \mapsto k \mapsto x \in l_net \land k \mapsto n \mapsto y \in l_net \land x \in sensors \land y \in sensors \Rightarrow$
$x \in SANet[\{n\}] \land y \in SANet[\{k\}] \land x \mapsto y \in closure(SNet)$
**@inv42** $\forall n, m \cdot n \mapsto m \in recovered_ANet \Rightarrow (\exists k \cdot k \in sensors \land n \mapsto m \mapsto k \in l_net)$

These relations describe links between nodes at a different level, hence they are disjoint from the actor links modelled by *ANet* (**@inv31 and @inv32**). *SNet* and *SANet* are also disjoint sets (**@inv33**). Moreover, they are symmetric and irreflexive sets (**@inv34-37**). We also require that, for each link $n \mapsto m$ in *SANet*, one of these nodes should be a sensor node and the other one should be an actor node (**@inv38**).

The next two invariants (**@inv39** and **@inv40**) express the property that every node involved in either *SNet* or *SANet* should be active. The invariant **@inv41** states that the actor–actor recovered links of *l_net* can be reconstructed by actor–sensor and sensor–sensor links contained correspondingly in *SANET* and *SNET*. Finally, the invariant **@inv42** is a gluing invariant showing how the global structure *recovered_ANet* can be replaced by the local information *l_net* and the available sensor nodes.

We add two new events for adding links between sensor nodes into *SNet* and links between sensor and actor nodes into *SANet*: **AddSLink** and **AddSALink**.

**AddSLink**
**ANY** n m **WHERE**
**@grd1** $n \in sensors \land m \in sensors$
**@grd2** $Status(n) = ok \land Status(m) = ok$
**@grd3** $n \mapsto m \notin SNet$
**@grd4** $n \neq m$
**@grd5** $FailedNodeNeigh = \varnothing$
**THEN**
**@act1** $SNet := SNet \cup \{n \mapsto m, m \mapsto n\}$
**END**

**AddSALink**
**ANY** n m **WHERE**
**@grd1** $n \in actors \land m \in sensors$
**@grd2** $Status(n) = ok \land Status(m) = ok$
**@grd3** $n \mapsto m \notin SANet$
**@grd5** $FailedNodeNeigh = \varnothing$
**THEN**
**@act1** $SANet := SANet \cup \{n \mapsto m, m \mapsto n\}$
**END**

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

The **AddSLink** event is similar to **AddLink** except for a different guard, which that models that, for every link $n \mapsto m$ added in **AddSLink**, $n$ and $m$ should be sensor nodes. The event **AddSALink** models adding of links between sensors and actors in a similar way.

The events **AddSLink** and **AddSALink** belong to the operational phase of the network, hence they also have the condition $FailedNodeNeigh = \varnothing$ as one of their guards (**@grd5** in both events).

```
DeactivateNode
refines DeactivateNode
THEN
  @act8 l_net := l_net \ (({n} × dom(ANet) × Status⁻¹[{ok}]) ∪
    (dom(ANet) × {n} × {n}) ∪ (dom(ANet) × {n} × dom(SNet)))
  @act9 SANet := {n} ⩤ SANet ⩥ {n}
END
```

In the previous models, removing an actor and all its connections was modelled by the event **DeactivateNode**. In this model we refine **DeactivateNode** because all the connections through actor and sensor nodes to and from a failed actor should be removed from *l_net* (**@act8**). The expressions $dom(ANet) \times \{n\} \times \{n\}$ and $dom(ANet) \times \{n\} \times dom(SNet)$ describe respectively all the direct and recovered links to $n$. These links belong to the relational expression $dom(ANet) \times \{n\} \times Status^{-1}[\{ok\}]$ that was used in the previous models. Therefore, action refinement is proven for **DeactivateNode** (the simulation proof obligation (*REF_SIM*)). Finally, we add a new action for updating *SANet* after removing an actor node (**@act9**).

A similar action refinement is done in the event **Addl_net2hoplink**. The action **@act1** in **Addl_net2hoplink** is refined by updating *l_net* with

$$l_net \setminus ((\{k\} \times \{n\} \times sensors) \cup (\{n\} \times \{k\} \times sensors)) \cup \{n \mapsto k \mapsto m, k \mapsto n \mapsto m\}.$$

This assignment actually restricts removing the recovered links through sensors from *l_net* when there is an alternative path through actors.

```
Recovery1
extends Recovery1
ANY n m k x y WHERE
  @grd5 x ∈ SANet[{n}] ∧ y ∈ SANet[{k}]
  @grd6 x ↦ y ∈ closure(SNet)
THEN
  @act5 l_net := (l_net \ ({n ↦ k ↦ m, k ↦ n ↦ m} ∪ indir(n, m) ∪ indir(k, m)))
    ∪ indir(k, n) ∪ indir ˜(n, k) ∪ indir(n, k) ∪ indir ˜(k, n)
    ∪ {n ↦ k ↦ x, k ↦ n ↦ y}
END
```

The event **Recovery1**, which models the recovery after an actor failure, is refined based on the information present in *SNet* and *SANet*. Compared to the previous version of the event, there are two additional parameters $x$ and $y$ representing the sensor nodes connected with the respective actor nodes $n$ and $k$ (**@grd5**). Also, $x$ and $y$ have either a direct or indirect link between each other, via $closure(SNet)$ (**@grd6**).

The action **@act5** in **Recovery1** was described non-deterministically in the previous model. We now refine this assignment to a deterministic one. Specifically, we replace $\{k\} \times \{n\} \times Status^{-1}[\{ok\}] \setminus ANet[\{n\}]$ with $k \mapsto n \mapsto y$ and, similarly, $\{n\} \times \{k\} \times Status^{-1}[\{ok\}] \setminus ANet[\{n\}]$ with $n \mapsto k \mapsto x$. The similar action **@act4** in **Recovery2-2** is refined accordingly. These actions correspond to line 20 of the recovery algorithm pseudo-code shown in Fig. 4.

To show the correctness of the modifications made in **Recovery1**, the simulation proof obligation (REF_SIM) is again discharged. This ensures that the execution of the concrete version of **Recovery1** is not contradictory to that of the abstract one. The obligation is proved by showing that the abstract event models a more general case than the concrete one.

In this refined model, we uncover the sensor infrastructure and employ it for the actor recovery. This model is a refinement of the previous models, respecting all the introduced invariants. The third model illustrates the usage of sensors as a fault tolerance mechanism for the actor coordination. This refinement step includes a large number of proof obligations because, firstly, we add a new type of nodes and their corresponding networks and, secondly, all the non-deterministic actions in the previous model are substituted with deterministic ones. The latter leads to the simulation proof obligations that are generated to prove the consistency between the abstract and concrete models.

## 7. Third refinement: sensor-based recovery via the shortest actor path

Our previous system model re-establishes connections through sensor nodes between pairs of actor nodes that are direct neighbours of a failed actor node. However, this is not an optimal mechanism since the actor nodes can be far from each other and may need to involve numerous sensor nodes to re-establish the connection, while there might be a shorter path for this.

```
INVARIANTS
@inv43 locX ∈ NODE → 0..X
@inv44 locY ∈ NODE → 0..Y
...
```

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

To determine the shortest path between actor nodes, we need information about the physical location of the nodes. For this purpose, in this model we introduce new variables *locX* and *locY* to store the $(x, y)$ coordinates for actor nodes (**@inv43, @inv44**).

**CONSTANTS** *X Y init_locX init_locY r_s r_a dist*
**AXIOMS**
**@axm20** $X \in \mathbb{N}1$
**@axm21** $Y \in \mathbb{N}1$
**@axm22** $init_locX \in NODE \rightarrow 0..X$
**@axm23** $init_locY \in NODE \rightarrow 0..Y$
**@axm24** $dist \in NODE \times NODE \rightarrow \mathbb{N}1$
**@axm25** $r_a \in \mathbb{N}1$
**@axm26** $r_s \in \mathbb{N}1$

A number of new constants are also added into the model context component. Namely, constants $X$ (**@axm20**) and $Y$ (**@axm21**) represent the height and width of the field where the WSAN has been deployed. The locations of all nodes are initialised with constants *init_locX* (**@axm22**) and *init_locY* (**@axm23**). We also define a new constant *dist* (**@axm24**) as a function on node pairs in such a way that the distance $dist(i, j)$ stands for

$$(locX(i) - locX(j)) * (locX(i) - locX(j)) + (locY(i) - locY(j)) * (locY(i) - locY(j))$$

As explained in Section 3, a link between nodes can be added only when they are within the communication range of each other. To model that, we introduce the actor communication range *r_a* (**@axm25**) and the sensor communication range *r_s* (**@axm26**) as abstract constants in the model context.

**INVARIANTS**
...
**@inv45** $\forall n, m \cdot n \mapsto m \mapsto m \in l_net \Rightarrow dist(n \mapsto m) \leq r_a$
**@inv46** $\forall n, m \cdot n \mapsto m \in SNet \Rightarrow dist(n \mapsto m) \leq r_s$
**@inv47** $\forall n, m \cdot n \mapsto m \in SANet \Rightarrow dist(n \mapsto m) \leq r_s$
**@inv48** $\forall n, m, k \cdot n \mapsto m \mapsto k \in l_net \wedge k \in actors \Rightarrow dist(n \mapsto k) \leq r_a \wedge dist(m \mapsto k) \leq r_a$
**@inv50** $\forall n, m \cdot n \in FailedNodeNeigh \wedge m \in FailedNodeNeigh \Rightarrow dist(n \mapsto m) \leq (r_a + r_a)$
...

Having defined the communication range and the distance function, we can now formulate the network requirements for adding links by **@inv45-50**. These invariants state that a link between two nodes in *ANet*, *l_net*, *SNet* and *SANet* can be added only if the nodes are within the pre-defined range from each other.

In the refined model, the guards of the event **AddLink** are strengthened to permit adding links only when the nodes are within the range. Specifically, the event is enabled when the physical distance between nodes is less than or equal to the communication range of actor nodes ($dist\ (n \mapsto m) \leq r_a$). Similarly, the guards of the events **AddSLink** and **AddSALink** are strengthened to add links between sensors and sensors–actors in *SNet* and *SANet* only if the distance between them is less than or equal to *r_s* ($dist(n \mapsto m) \leq r_s$).

The network recovery modelled so far proceeds by choosing one of the neighbours of a failed node and re-establishing its connection to another neighbour. The choice of a pair of the neighbour nodes is completely arbitrary. To optimise the recovery process, we introduce some node priorities when selecting a (first) node for recovery. Such a node priority stands for the number of direct (1-hop) neighbours of the node and can also be referred to as the *node degree*.

Once the first actor node is selected according to its degree, the choice of the second one depends on the physical location information of the network nodes introduced in this refinement step. More specifically, the closest actor node (according to the function *dist* defined above) is chosen to re-establish connection with.

As explained in Section 2.2, a direct neighbour of the failed node with the highest degree calculates its distance from other direct neighbours of the failed node and chooses to connect (via sensors) to the closest actor. Next, an actor node with the second highest degree calculates its distance to the other neighbours of the failed actor. This process continues until all the neighbours of the failed actor have a new route via sensors.

**INVARIANTS**
...
**@inv51** $degree \in rec_set \nrightarrow 0..card(NODE)$
**@inv52** $Neigh_temp \subseteq FailedNodeNeigh$
**@inv53** $Neigh_temp \cap dom(degree) = \varnothing$
**@inv54** $Neigh_temp = \varnothing \Rightarrow FailedNodeNeigh = dom(degree)$

The function variable *degree* is introduced to store the current degree of each neighbour of the failed actor (**@inv51**). The degree of all the neighbours of the failed node has to be recalculated every time the recovery starts. The neighbour nodes that still require to recalculate their degree are stored in a set variable *Neigh_temp* (**@inv52-54**). Essentially, this variable controls the loop of degree calculation, while the natural number expression *card*(*Neigh_temp*) also serves as a loop variant guaranteeing its termination.

The invariant **@inv53** states that, when the calculation of the degree is finished, the calculated degree is assigned to all the neighbours of the failed node. When *Neigh_temp* becomes empty, the computation of *degree* is completed, establishing the invariants **@inv52** and **@inv54**.

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

ARTICLE IN PRESS


```
DeactivateNode
extends DeactivateNode
THEN
  @act7 Neigh_temp := dom(l_net⁻¹[{n}])
END
```

```
Degree
ANY n WHERE
  @grd1 FailedNodeNeigh ≠ ∅
  @grd2 n ∈ Neigh_temp
THEN
  @act1 degree :=
        degree ∪ {n ↦ card(dom(l_net⁻¹[{n}]))}
  @act2 Neigh_temp := Neigh_temp \ {n}
END
```

The event **DeactivateNode** needs to be extended to assign a new value to the variable *Neigh_temp* (**@act7**). When $n$ is the deactivated (failed) node, then $l_net^{-1}[\{n\}]$ denotes all its 1-hop neighbours.

In this refinement step we also add a new event **Degree** to model the degree calculation loop, i.e., to calculate the degree of all the actors in *Neigh_temp*. The event belongs to the recovery phase of the network, i.e., *FailedNodeNeigh* is not empty (**@grd1**). **Degree** recalculates the degree for one element of the set variable *Neigh_temp* and then removes it from the set. As mentioned above, the decreasing cardinality of *Neigh_temp* guarantees termination of the looping events modelling the recovery process.

To guarantee non-interference from the other events of the recovery phase, we block their execution until the degree calculation is completed. Blocking is done by adding the guard $Neigh_temp = \varnothing$.

```
Recovery1
extends Recovery1
WHERE
  @grd7 Neigh_temp = ∅
  @grd8 degree(n) = max(dom(degree⁻¹))
  @grd9 ∀i · i ∈ dom(degree) \ {n, k} ⇒ dist(n ↦ k) ≤ dist(n ↦ i)
THEN
  @act6 degree := {n} ⩤ degree
END
```

The recovery events **Recovery1**, **Recovery2-1** and **Recovery2-1** are also refined to constrain node selection for recovery. The presented event **Recovery1** contains three additional guards. The guard **@grd7** blocks the event during the node degree calculation. The guard **@grd8** requires the neighbour node $n$ to be one with the highest degree. Finally, the guard **@grd9** constrains selection of the node $k$ to be the closest neighbour of $n$. The additional action **@act6** removes the node $n$ from *degree* once the node is re-connected. The same guards and action are added to **Recovery2-1** and **Recovery2-2**.

The added new guards constrain the random choice of nodes from *FailedNodeNeigh* in the same way as in the original proposed recovery algorithm in [5]. In particular, **@grd8** corresponds to lines 16–17 in Fig. 4 and **@grd9** corresponds to lines 18–19 in Fig. 4.

## 8. Proving and refinement

Our aim throughout the development has been to make refinement steps in such a way that the proofs are manageable by the RODIN platform — our modelling and proving environment. Our initial model is sufficiently abstract to be refined for modelling some other recovery algorithms as well. To faithfully model and verify our target algorithm [5], we had to gradually introduce several features into our formal development, starting from the initial model:

1. the sensor network, the actor network, and the sensor–actor network;
2. the localised node knowledge;
3. the optimal nature of the recovery path.

In this paper we have chosen to add these features to the initial model in the order 2.,1.,3., in three refinement steps. This order of modelling the features produced a rather balanced automated-versus-interactive proving ratio in each refinement step. However, we did not try to introduce these features in any other order, so it remains as an informative future exercise to study a different order strategy.

We illustrate the proof statistics of our development in Table 2. These figures show the number of proof obligations generated by the RODIN platform as well as the number of obligations automatically discharged by the platform and those interactively proved. We notice that, in each refinement step, we have about the same percentage of automatically versus interactively proved, namely on average 55% automated proving and 45% interactive proving.

A high number of interactive proofs came from reasoning about set comprehension and unions. The interactive proving has also often involved manually suggesting values to discharge various properties containing logical disjunctions or existential quantifiers.

In order to prove the functional correctness properties given in the initial model, we have defined the predicate *connected* to express the existence of a path between a subset of nodes in a network. A number of useful properties of *connected* were postulated as model axioms, whose consistency has been proven within the external theorem prover HOL. This is because, at the moment, the provers of the RODIN platform are better suited for proving properties of particular model elements (e.g., invariant preservation or refinement of a specific event) than for proving general properties of recursive data structures.

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

**Table 2**
Proof statistics.

| Model | Number of proof Obligations | Automatically Discharged (%) | Interactively Discharged (%) |
|---|---|---|---|
| Context1 | 19 | 68 | 32 |
| Context2 | 4 | 100 | 0 |
| Initial Model | 104 | 72 | 28 |
| 1st Refinement | 40 | 55 | 45 |
| 2nd Refinement | 57 | 52 | 48 |
| 3rd Refinement | 69 | 58 | 42 |
| Total | 293 | 63 | 37 |

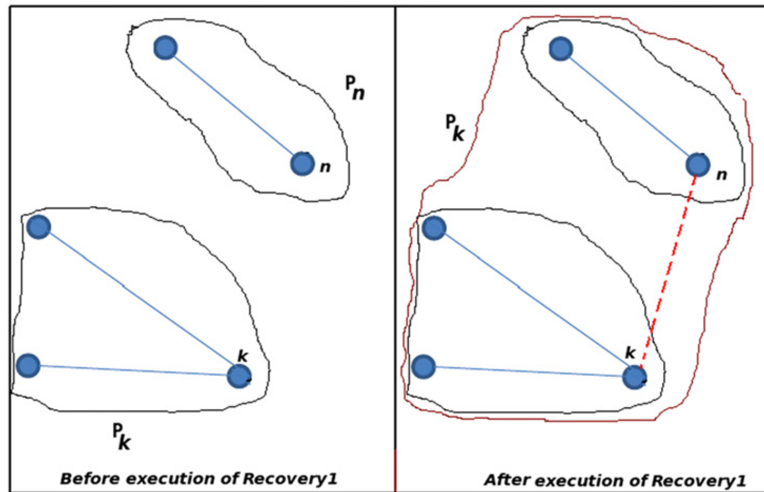



**Fig. 6.** Before and after tree structure of Recovery1 in the initial model.

We note that there are plans to bridge the RODIN platform with external provers such as Isabelle or HOL, exactly for the reason of proving the consistency of various model axioms using such external provers.

The most essential property for proving correctness of the recovery mechanism is the axiom **@axm14**. It states that each step of recovery reduces a number of disconnected partitions among the neighbours of a failed node. Specifically, adding links between two elements of a node forest allows one to merge those elements into one node tree and thus reduces the number of disconnected node partitions:

**@axm14** $\forall S1, S2, L1, L2, n, m \cdot n \in S1 \wedge m \in S2 \wedge connected(S1 \mapsto L1) = TRUE \wedge$
$connected(S2 \mapsto L2) = TRUE \Rightarrow connected((S1 \cup S2) \mapsto (L1 \cup L2 \cup \{n \mapsto m, m \mapsto n\})) = TRUE$

For each recovery event, we have proved that two separated partitions become connected and create a new tree by using the accordingly specialised **@axm14** as an hypothesis for the proof. For instance, to prove preservation of the invariant **@inv21** for the event **Recovery** of the initial model, we have split the generated proof obligation into two simple ones:

$(1) \forall p_1 \cdot p_1 \in dom(rec_forest) \Rightarrow connected((\{p_k\} \times rec_forest[\{p_n\}])[\{p_1\}] \mapsto rec_links) = TRUE$
$(2) \forall p_1 \cdot p_1 \in dom(rec_forest) \Rightarrow connected((\{p_n\} \lhd rec_forest)[\{p_1\}] \mapsto rec_links) = TRUE$

To prove these proof obligations, we have specialised the universally quantified variables $S1, S2, L1, L2, n, m$ in the axiom **@axm14** by the values $(\{p_k\} \times rec_forest[\{p_n\}])[\{p_1\}]$, $(\{p_n\} \lhd rec_forest)[\{p_1\}]$, $rec_links$, $rec_links$, $n$ and $k$, respectively. As a result, we have proved that after merging two separated partitions $p_n$ and $p_k$ (illustrated in Fig. 6), we have only one connected partition consisting of $p_n$ and $p_k$.

## 9. Discussion: pattern-driven formal development

In this section we discuss how we can generalise and reuse the formal development of self-recovering sensor–actor networks presented so far. To achieve this, we rely on the idea of *formal patterns*, which may be identified and then repeatedly applied to facilitate the refinement process.

We call such formal patterns *refinement patterns*. These patterns generalise certain typical model transformations reoccurring in a particular development method. They can be thought of as refinement rules in large.

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

ARTICLE IN PRESS

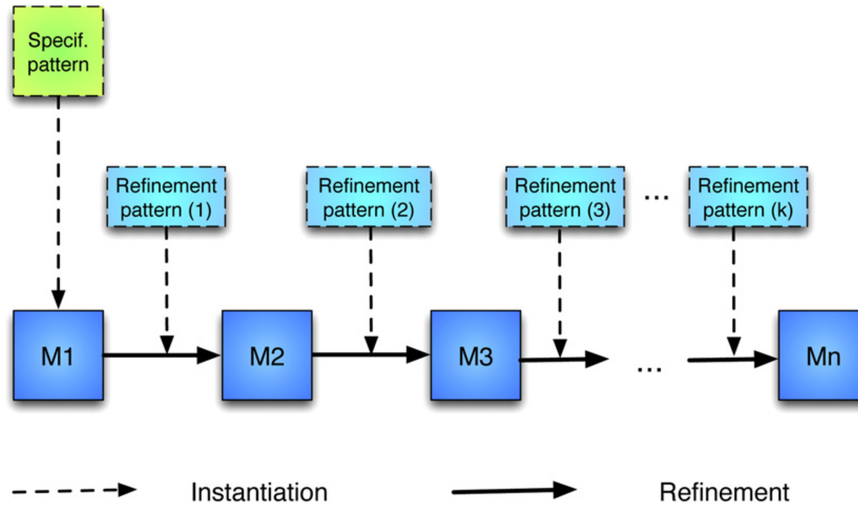




**Fig. 7.** Pattern-driven model development.

The application of refinement patterns is compositional. Hence, some large model transformation steps can be represented by a certain combination of refinement patterns, and therefore can also be seen as refinement patterns per se. Moreover, reducing the execution of a refinement step to a number of syntactic manipulations over a model provides a basis for automation. Finally, such an approach can potentially support reuse of not only models but also proofs. Indeed, by proving that an application of a generic pattern produces a valid refinement of a generic model, we at the same time verify correctness of such a transformation for any of its instances.

Graphically, pattern-driven formal development can be represented as shown in Fig. 7. The initial model $M_1$, the starting point of the development, is created by instantiating a special template, called *a specification pattern*. Essentially, it is a parametrised specification (e.g., an Event-B model). During pattern instantiation, the model parameters are substituted with concrete data structures, while the model variables and events can be renamed. The model constraints (e.g., Event-B axioms) given for these parameters become the theorems to be proved to show that this pattern is applicable for the given concrete values. If the instantiation succeeds, the model invariant properties (together with their proofs) are obtained for free, i.e., without any proofs. For more details about formal pattern-driven development in the Event-B framework, see [13].

The development presented in Sections 4–7 formalises the specific recovery algorithm that can be used to restore communication links in sensor–actor networks. It would be desirable, if possible, to generalise this approach to a wider class of networks, identifying at the same time typical design steps that can be then formalised as refinement patterns. In this section we put forward the initial steps towards achieving this goal.

We start by presenting a specification pattern that provides us with a suitable abstract model, which will be the starting point of a formal development by refinement. We define basic data structures and operations for any network containing communicating nodes. Specifically, we construct a node graph *G(NodeType1, Net1)* by defining *NodeType1* as a subset of all nodes *NODE* and *Net1* as the set of all the direct communication links between *NodeType1* nodes. The sets *NODE* and *NodeType1* are defined in the model context, while *Net1* is a model variable. The node status reflecting whether it is active or failed is stored in a function variable *Status*. The initial status of the network is given by a constant *Initial_status*.

The context for our specification pattern is as follows.

**CONTEXT** *Data*
**CONSTANTS** *active failed Init_status*
**SETS** *NODE NodeType*1 *NODE_STATUS*
**AXIOMS**
  **@axm1** $finite(NODE)$
  **@axm2** $NodeType1 \subseteq NODE$
  **@axm3** $NODE \neq \varnothing$
  **@axm4** $partition(NODE_STATUS, \{active\}, \{failed\})$
  **@axm5** $Init_status \in NODE \rightarrow NODE_STATUS$
**END**

The sets and constants defined here are the specification pattern parameters, which can be instantiated during pattern application. Many concrete details, such as the type of nodes, the communication ranges for different node types, specific details of communication recovery and so on are missing in this model and will be later introduced in dedicated refinement steps (refinement patterns).

In the machine part of our initial model we have six invariants as shown below. The dynamic status of each node (active or failed) is modelled by a function *Status* (**@inv1**). A relation *Net*1 denotes the bidirectional, non-failed node links (**@inv2** and **@inv3**). This relation is irreflexive (**@inv4**) and symmetric (**@inv5**). In addition, the recovered links are modelled by a relation *Coord_Net*1 (**@inv6**) that denotes those links which are not included in the relation *Net*1. These invariants express the expected network properties that should be preserved.

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

```
INVARIANTS
    @inv1 Status ∈ NODE → NODE_STATUS
    @inv2 Net1 ∈ NodeType1 ↔ NodeType1
    @inv3 ∀n, m · n ↦ m ∈ Net1 ⇒ Status(n) = active ∧ Status(m) = active
    @inv4 NodeType1 ◁ id ∩ Net1 = ∅
    @inv5 Net1 = Net1⁻¹
    @inv5 Coord_Net1 ∈ NodeType1 ↔ NodeType1
```

The **initialisation** event sets the status of all the nodes according to the given parameter *Init_status*, while the *Net*1 and *Coord_Net*1 relations are initially empty.

The model contains four event operations, which are abstracted versions of these events presented in Section 4. In the **AddNode** event, a new node is added by updating the function *Status*.

```
initialisation
BEGIN
  @act1 Status := Init_status
  @act2 Net1 := ∅
  @act3 Coord_Net1 := ∅
END
```

```
AddNode
ANY n WHERE
  @grd1 n ∈ NodeType1
  @grd2 Status(n) = failed
THEN
  @act1 Status(n) := active
END
```

In the **AddLink** event we non-deterministically add a link in both directions. In later refinement steps we are going to restrict adding communication links based on the node communication range.

The **RemoveNode** event changes the status of a node from *active* to *failed*. Also, all the links of that node (in both directions) are removed from *Net*1 and *Coord_Net*1. Again, this is needed to preserve the **@inv3** proof obligation.

```
AddLink
ANY n m WHERE
  @grd1 n ∈ NodeType1 ∧ m ∈ NodeType1
  @grd2 Status(n) = active
          ∧Status(m) = active
  @grd3 n ≠ m
  @grd4 n ↦ m ∉ Net1
THEN
  @act1 Net1 := Net1 ∪ {n ↦ m, m ↦ n}
END
```

```
RemoveNode
ANY n WHERE
  @grd1 n ∈ NodeType1
  @grd2 Status(n) = active
THEN
  @act1 Status(n) := failed
  @act2 Net1 := {n} ⩤ Net1 ⩥ {n}
  @act3 Coord_Net1 := {n} ⩤ Coord_Net1 ⩥ {n}
END
```

Finally, the **Recovery** event adds a link in both directions, after a link is removed from the network which leads to its partitioning.

```
Recovery
ANY n m WHERE
    @grd1 n ∈ NodeType1 ∧ m ∈ NodeType1
    @grd2 Status(n) = active ∧ Status(m) = active
    @grd3 n ≠ m
    @grd4 n ↦ m ∉ Net1
THEN
    @act1 Coord_Net1 :∈ Coord_Net1 ∪ {n ↦ m, m ↦ n}
END
```

By instantiating this specification pattern, we can obtain an abstract network model that can serve as the starting point of formal development. For instance, if we instantiate *NODE_STATUS* with the concrete type *STATUS*, *Init_status* with the concrete constant *initial_node_status*, *active* and *failed* with the concrete constants *ok* and *fail*, and rename the variables *Net*1 by *ANet* and *Coord_Net*1 by *recovered_ANet*, we obtain the initial abstract model for our formal development described in Sections 4–7. The network model presented in Section 4 then becomes a refinement of this instantiated model. The refinement is easily proved since both context and machine shown in Section 4 are essentially extensions, only adding new data structures and variables as well as a couple of new events for modelling communication recovery defined on these new variables.

Below we discuss some typical transformation (refinement) steps for formal network models such as the one described above:

- **Introduction of specific types of network nodes (e.g., sensors and actors) and their interdependencies.** Here we should distinguish between two different development strategies resulting in distinct refinement patterns. We can gradually introduce different types of nodes (different sub-networks) by adding new data structures, variables and events (so called superposition refinement) and tying them together by new axioms and invariants. That approach is taken in this paper. Alternatively, we can start with a complete network consisting of abstract nodes and then introduce a node classification into different types. The events of the abstract model modelling abstract network functionality (like adding new nodes or links between them) would be then refined by several more concrete versions taking into account what types of nodes are involved. In both approaches, the applicability conditions of the corresponding patterns would reflect which development strategy is chosen. In Fig. 8 we show the application of a refinement pattern to transform the initial model to a more concrete one;

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

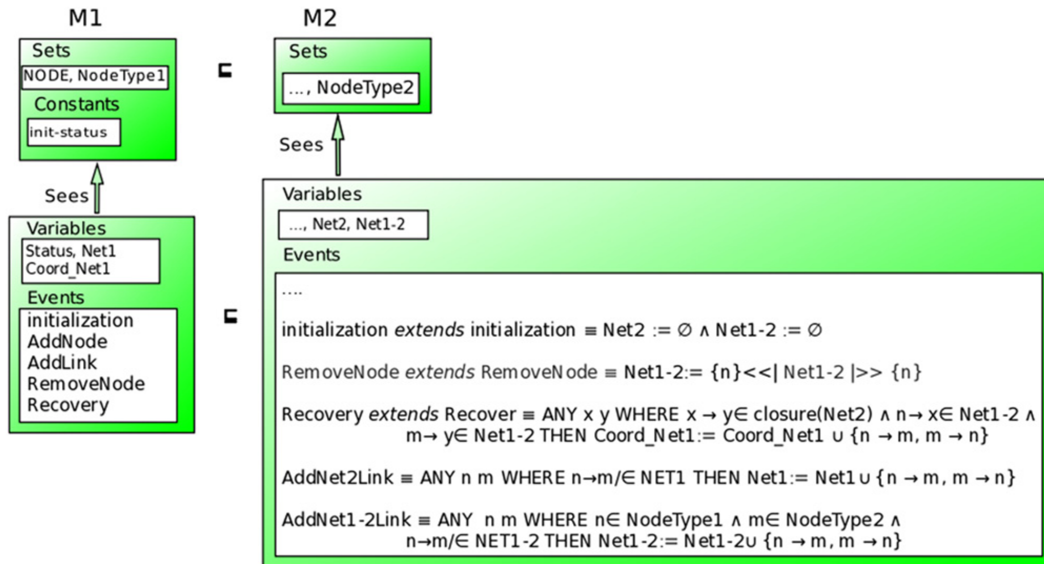



**Fig. 8.** Application of the refinement pattern to transform M1 to M2.

- **Refinement of the communication recovery mechanisms affecting a particular (localised) part of the network.** Essentially, such patterns would be parametrised by data structures defining the network area and the types of nodes involved in recovery. For a refinement pattern to be applied, the input model would need to already have the event(s) modelling recovery in an abstract way. These events will be directly affected by pattern application. Moreover, new variables and events to model intermediate or temporary results of recovery would be also added;
- **Introduction of physical attributes or characteristics (for instance, communication distance) for different types of nodes.** This information is used to restrict different network operations (e.g. adding communication links between nodes) by checking that the involved network elements satisfy the given constraints. The corresponding event guards are added, while some events may be split to account for different cases. Care should be taken to ensure that the pattern application does not introduce deadlocks in the resulting model or, e.g., prevent network recovery mechanisms from achieving their goals.

Some of these refinement steps can be merged or applied in a different order. Moreover, it would be unrealistic to expect to express all of these steps by single refinement patterns. Different network types may, and probably will, require to create separate patterns to account for specific differences between them. To investigate different cases of communicating networks and, as a result, build an extensive collection of such patterns will be a part of our future work.

## 10. Related work

In this section we consider related contributions with respect to three topics: firstly, we motivate our choice of formalism; secondly, we discuss other formal approaches to modelling various aspects of wireless networks; and thirdly, we consider contributions related to the pattern-based idea of reuse that we explore in our paper.

*Choice of formalism.* Choosing a modelling formalism is an essential step for formalisation of any system. The main reason for choosing Event-B in our work is due to a global modelling and reasoning approach that Event-B provides. Event-B has evolved from the B-Method [8] and the Action Systems [14] formalisms. The latter formalism was introduced in 1983 for modelling parallel and distributed algorithms from a *global perspective*. Process algebra formalisms such as CSP [15] or CCS [16] focus on the particular (sequential) computation that each process specifies, possibly also synchronising with other processes. In contrast, Action Systems and, later on, Event-B promote the so-called *system approach* [17]. Following the system approach, we can reason about global properties of the system, without having to consider the properties of sequential process computations and their communications. Moreover, we can add details at the global level, thus developing the system in a clean top-down approach. Because we focus on the global view of the system to be modelled, we initially specify a very abstract model, on top of which we identify details to add in subsequent refinement steps.

Another feature that we strongly appreciate about Event-B is its tool support provided by the RODIN platform [7,9,10,18]. While we can prove theorems in various theorem provers such as HOL [19], PVS [20], or Isabelle [21], RODIN comes with an integrated approach for modelling *and* proving in an interleaving manner, as we have discussed in Section 2.1. While other state-based approaches such as Z [22] and Abstract State Machines [23] propose refinement methodologies, in the RODIN platform we construct a complete system model and then the platform generates and partially or completely discharges the required proof obligations. The system approach is also supported and highly emphasised in the RODIN platform. For instance, if one needs to specify several components communicating with each other, one starts from a global view containing all the components in one model and then *decomposes* this model into components [6,24,25].

*Other formal approaches.* In this paper we have employed Event-B in order to formalise a recovery algorithm for wireless sensor–actor networks. There are many other formalisms used for modelling aspects of WSANs. A recent experiment [26] shows that several formalisms are well-suited for modelling and analysing various aspects of sensor networks. The authors of [26] conclude that a new formalism can be defined that combines the advantages of at least three other formalisms, including Ambient Calculus [27] and a timed extension of the Object-Z [28]. In [29], the authors show that combining model checking and refinement proofs is advantageous for verifying performance and correctness of protocols for wireless networks. Model checking is also used in [30], to model and verify a wireless sensor network protocol, while in [31] the authors employ Petri Nets [32] to verify the design and structural properties of wireless sensor and actuator networks. Wireless sensor and actuator networks are networks containing a control system whose sensors and actuators communicate with each other via a wireless network. We note in this context that we model actors instead of actuators: a sensor–actuator pair is typically related to a control loop, while a sensor–actor pair is a coordination pair. Our actors are in control of the entire WSAN and form a subnetwork of their own, whose recovery we address.

Some contributions in the domain of refinement-driven development focus on modelling and analysing network protocols in a static environment, i.e., without considering activation and deactivation of nodes. In [33,34], a formal development of a distributed leader election protocol on a connected network graph is presented, considering several safety properties. In [6], the development of a routing algorithm for mobile agents is modelled. While all the nodes and links are predefined and fixed, the networks operate in a decentralised manner, similarly to the distributed nature of recovery in our model.

In [35], the authors present a discovery algorithm for developing the network topology. The topology discovery is an issue relevant to most routing algorithms. All the nodes in a network need to discover and maintain information on the network topology, so that they have to cooperate to establish connections. The development guarantees that at the end of the discovery algorithm, all the nodes in the network have the same information about the network topology. However, the access to this type of information about the whole network is not possible in a dynamic network where nodes are randomly added and removed. Another correct-by-construction distributed development is shown in [36], where the authors present a formal development of a distributed reference counting algorithm. Both works focus on preservation of the consistency properties of the relevant protocols.

In [37,38], a modelling and analysing approach for dynamic decentralised systems is proposed. The author applies a stepwise and event-based approach to specify dynamic behaviours of a system that has no dedicated network infrastructure. As an application of the method, a mobile ad-hoc network (MANET) is modelled as an abstract model. In the next model, by considering the management of the IP addresses of nodes and exchange messages, the routing algorithm is refined. The initial model is composed of two main parts: the structure of the MANET and the routing protocol. The similarity of our approach to this paper is that we both choose the same strategy of separating the modelling of the network structure from the algorithms that employ these structures.

*Refinement and reuse.* The refinement pattern of our work was inspired by several contributions on automation of the refinement process. The Refinement Calculator tool [39] has been developed to support program development using the Refinement Calculus theory [40] The theory has been formalised in the HOL theorem prover [19], while specific refinement rules were proved as HOL theorems. A similar framework consisting of refinement rules (called tactics) and the tool support for their application has been developed by Oliveira, Cavalcanti, and Woodcock [41]. The framework (called ArcAngel) provides support for another version of the Refinement Calculus [42]. The obvious disadvantage of both these frameworks is that the refinement rules that can be applied usually describe quite small, localised transformations.

In [43,44] a design pattern within the framework of Event-B is proposed as a template on how to solve a problem which can be used in many different situations. The idea is to have a number of pre-defined solutions that cooperate in a development with some modifications or instantiations. The authors focus on instantiating generic models (specification patterns) to gradually develop a system model. On the other hand, our approach relies on the notion of refinement patterns as automatic parametrised model transformations, where the resulting model is a refinement of the input model. Another experiment on employing the idea of refinement patterns appears in [45,46], where it is applied for formal development of Network-on-Chip routing algorithms.

## 11. Conclusions

In this paper, we have formalised a distributed recovery algorithm that was introduced and simulated in [5]. The algorithm addresses the network partitioning problem in WSANs generated by actor failures. We have modelled the algorithm and the corresponding WSANs in Event-B. One part of our contribution here consists in proving correctness and successful termination of the recovery algorithm. More importantly, we set up a formal model of arbitrary WSANs that can evolve dynamically by adding nodes and their corresponding links as well as by removing nodes and their links. We have specified the network structure and the recovery mechanism at four increasing levels of abstraction that refine each other. We have formally proved the refinement steps using the theorem prover tool RODIN [7,9,10,18].

One of the most interesting aspects put forward is formal modelling and development of actor coordination links that can be seen in three forms: a direct actor–actor link, an indirect, not further specified path, or an indirect path through sensor nodes. We have gradually developed these cases by refinement, specifically, by replacing the first form with the second

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

form, and then by the third one. However, the proved refinement relation between them shows that all three forms can be present in a network and thus provide various coordination alternatives for actors. In this respect, one can define different coordination classes, e.g., for delegating the most security-sensitive coordination to the direct actor–actor coordination links, the least real-time constrained coordination to indirect links, and the safety-critical coordination to both direct actor links and indirect sensor paths between actors. This contributes to modelling fault-recovery in wireless networks.

Specifically, we have formalised the direct actor–actor recovery that relies on the global network information, which can be formally represented as the relation *ANet*; *ANet*. Moreover, we have successfully proved correctness and termination of this formalised coordination recovery. In the next two refinement steps, we have specified indirect actor–actor recovery via arbitrary nodes or via sensors. In these two cases, we do not need the global network information to perform the recovery but rely instead on the information stored locally by the *l_net* relation. Since these two forms of indirect recovery are correct refinements of the direct actor–actor recovery, we can deduce that the distributed recovery is also correct and successfully terminating. Finally, the last refinement step optimises the recovery process by taking into account the physical location information of the network.

Using the sensor infrastructure as a temporary backup for actor coordination also aligns with the growing *sustainability* research of using resources without depleting them. Upon detecting a direct actor–actor coordination link between two actor nodes, all the sensor nodes contributing to a communication link between these actor nodes are released of their backup task, as illustrated in the events adding links between nodes in our models.

As a result, our formal development achieves a complete formalisation of the original algorithm presented in [5]. Our model presented in this paper also demonstrates the power and applicability of a formal refinement approach. The original algorithm in [5] consists only of the fourth recovery form, with actors, sensors, the *l_net* relation as well as the physical distance information. While this form is quite complex to model and verify, we have shown how to start from a more abstract version and prove its functional correctness, termination and other properties. Stepwise refinement of this initial model has allowed us to gradually handle the rising complexity while keeping the desired correctness properties preserved.

A more general message suggested by the achieved results of this paper is that they can be applied to any network with two coordinating categories of node, some more powerful than the others. The algorithm we have modelled is essentially a general one that can be reused as the basis for more complex networks. Indeed, in Section 9 we have provided a specification pattern that can be reused in similar situations. We have also discussed there possible refinement patterns for modelling dynamic networks with heterogeneous nodes.

## Acknowledgements

M. Kamali's work is partially supported by Nokia. This work is supported by IST FP7 DEPLOY project. We would like to kindly thank the reviewers for their detailed and pertinent comments on our paper.

## References

[1] I.F. Akyildiz, I.H. Kasimoglu, Wireless sensor and actor networks: research challenges, Ad Hoc Networks Journal 2 (4) (2004) 351–367.
[2] T. Melodia, D. Pompili, V.C. Gungor, I.F. Akyildiz, Communication and coordination in wireless sensor and actor networks, Proceedings of IEEE Transactions on Mobile Computing 6 (10) (2007) 1116–1129.
[3] K. Akkaya, M. Younis, Coverage and latency aware actor placement mechanisms in WSANs, International Journal of Sensor Networks 3 (3) (2008) 152–164.
[4] A. Abbasi, K. Akkaya, M. Younis, A distributed connectivity restoration algorithm in wireless sensor and actor networks, in: Proceedings of 32nd IEEE Conference on Local Computer Networks, LCN'07, IEEE, 2007, pp. 496–503.
[5] M. Kamali, S. Sedighian, M. Sharifi, A distributed recovery mechanism for actor–actor connectivity in wireless sensor actor networks, in: Proceedings of IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing, ISSNIP'08, IEEE, 2008, pp. 183–188.
[6] J.-R. Abrial, Modelling in Event-B: System and Software Design, Cambridge University Press, 2010.
[7] J.-R. Abrial, A system development process with Event-B and the RODIN platform, in: Proceedings of International Conference on Formal Engineering Methods, ICFEM'07, in: Lecture Notes in Computer Science, vol. 4789, Springer-Verlag, 2007, pp. 1–3.
[8] J.-R. Abrial, The B-Book: Assigning Programs to Meanings, Cambridge University Press, 1996.
[9] RODIN tool platform, http://www.event-b.org/platform.html.
[10] Rigorous Open Development Environment for Complex Systems (RODIN), IST FP6 STREP project, online at http://rodin.cs.ncl.ac.uk/.
[11] D. Craigen, S. Gerhart, T. Ralson, Case study: Paris Metro signaling system, in: Proceedings of IEEE Software, IEEE, 1994, pp. 32–35.
[12] J.N. Al-Karaki, A.E. Kamal, Routing techniques in wireless sensor networks: a survey, IEEE Wireless Communications Journal 11 (2004) 6–28.
[13] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, Patterns for refinement automation, in: Post-proceedings of FMCO 2009, Symposium on Formal Methods for Components and Objects 2009, in: Lecture Notes for Computer Science, Springer-Verlag, 2010, pp. 70–88.
[14] R.J. Back, R. Kurki-Suonio, Decentralization of process nets with centralized control, in: Proceedings of the 2nd ACM SIGACT-SIGOPS Symposium on 43 Principles of Distributed Computing, ACM, 1983, pp. 131–142.
[15] C.A.R. Hoare, Communicating sequential processes, Communications of the ACM Journal 21 (8) (1978) 666–677.
[16] R. Milner, A calculus of communicating systems, in: Lecture Notes in Computer Science, vol. 92, Springer-Verlag, 1980.
[17] M. Butler, E. Sekerinski, K. Sere, An action system approach to the steam boiler problem, in: Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control, in: Lecture Notes in Computer Science, vol. 1165, Springer-Verlag, 1996, pp. 129–148.
[18] J.-R. Abrial, M. Butler, S. Hallerstede, T.S. Hoang, F. Mehta, L. Voisin, Rodin: an open toolset for modelling and reasoning in Event-B, International Journal on Software Tools for Technology Transfer (STTT) (6) (2010) 447–466.
[19] HOL proof assistant, http://hol.sourceforge.net/.
[20] PVS specification and verification system, http://pvs.csl.sri.com/.
[21] Isabelle proof assistant, http://www.cl.cam.ac.uk/research/hvg/Isabelle/.
[22] M. Spivey, The Z Notation: A Reference Manual, second ed., in: Prentice Hall International Series in Computer Science, 1992.
[23] E. Borger, R. Stark, Abstract State Machines: A Method for High-Level System Design and Analysis, Springer, 2003.

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

[24] M. Butler, Decomposition structures for Event-B, in: Proceedings of Integrated Formal Methods, IFM'09, in: Lecture Notes in Computer Science, vol. 5423, Springer-Verlag, 2009, pp. 20–38.
[25] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, T. Latvala, Supporting reuse in Event B development: modularisation approach, in: Proceedings of Abstract State Machines, Alloy, B, and Z (ABZ 2010), in: Lecture Notes in Computer Science, vol. 5977, Springer-Verlag, 2010, pp. 174–188.
[26] J.S. Dong, K. Taguchi, X. Zhang, Specifying and verifying sensor networks: an experiment of formal methods, in: Proceedings of the 10th International Conference on Formal Engineering Methods, ICFEM 2008, in: Lecture Notes in Computer Science, vol. 5256, Springer, 2008, pp. 318–337.
[27] L. Cardelli, Abstractions formobile computation, in: J. Vitek, C. Jensen (Eds.), Secure Internet Programming: Security Issues forMobile and Distributed Objects, in: Lecture Notes in Computer Science, vol. 1603, Springer-Verlag, 1999, pp. 51–94.
[28] G. Smith, The Object-Z Specification Language, Kluwer Academic Publishers, 2000.
[29] A. McIver, A. Fehnker, Formal techniques for the analysis of wireless networks, in: Proceedings of the 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, ISoLA'06, IEEE Computer Society, 2006, pp. 263–270.
[30] A. Fehnker, L.V. Hoesel, A. Mader, Modelling and verification of the LMAC protocol for wireless sensor networks, in: Proceedings of the Integrated Formal Methods Conference, IFM'07, in: Lecture Notes in Computer Science, vol. 4591, Springer-Verlag, 2007, pp. 253–272.
[31] D. Martinez, A. Gonzalez, F. Blanes, R Aquino, J. Simo, A. Crespo, Formal specification and design techniques for wireless sensor and actuator networks, in: Sensors'11, vol. 11, MDPI Open Access, 2011, pp. 1059–1077.
[32] http://www.petrinets.info/.
[33] J.-R. Abrial, D. Cansell, D. Méry, A mechanically proved and incremental development of IEEE 1394 tree identify protocol, Formal Aspects of Computing Journal 14 (3) (2003) 215–227.
[34] J. Rehm, Proved development of the real-time properties of the IEEE 1394 root contention protocol with the Event-B method, International Journal on Software Tools for Technology Transfer 12 (2010) 39–51.
[35] T.S. Hoang, H. Kuruma, D. Basin, J.-R. Abrial, Developing topology discovery in Event-B, Science of Computer Programming Journal 74 (2009) 879–899.
[36] D. Méry, D. Cansell, Formal and incremental construction of distributed algorithms: on the distributed reference counting algorithm, Theoretical Computer Science Journal 364 (2006) 318–337.
[37] C. Attiogbé, Modelling and analysing dynamic decentralised systems, in: Proceedings of the 15th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC'09, IEEE, 2009.
[38] C. Attiogbé, Event-based approach to modelling dynamic architecture: application to mobile ad-hoc network, in: Proceedings of the 3rd International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA'08, in: Communications in Computer and Information 14 Science, vol. 17, Springer, 2008, pp. 769–781.
[39] M. Butler, J. Grundy, T. Løangbacka, R. Ruksenas, J.V. Wright, The refinement calculator: proof support for program refinement, in: Proceedings of Formal Methods Pacific, Springer-Verlag, 1997, pp. 40–61.
[40] R. Back, J.V. Wright, Refinement Calculus: A Systematic Introduction, Springer, 1998.
[41] M. Oliveira, A. Cavalcanti, J. Woodcock, Arcangel: a tactic language for refinement, Formal Aspects of Computing Journal (2003) 28–47.
[42] Carroll Morgan, Programming from Specifications, second ed., Prentice Hall, 1998.
[43] T.S. Hoang, A. Fürst, J.-R. Abrial, Event-B patterns and their tool support, in: Proceedings of the 7th IEEE International Conference on Software Engineering and Formal Methods, SEFM'09, IEEE, 2009, pp. 210–219.
[44] J.-R. Abrial, T.S. Hoang, Using design patterns in formal methods: an Event-B approach, in: Proceedings of the 5th International Colloquium, ICTAC'08, in: Lecture Notes in Computer Science, vol. 5160, Springer-Verlag, 2008, pp. 1–2.
[45] M. Kamali, L. Petre, K. Sere, M. Daneshtalab, Refinement-based modelling of 3D NoCs, in: Proceedings of the 4th International Conference on Fundamentals of Software Engineering, FSEN'11, in: Lecture Notes in Computer Science, vol. 7141, Springer-Verlag, 2012, pp. 236–252.
[46] M. Kamali, L. Petre, K. Sere, M. Daneshtalab, Formal modelling of multicast communication in 3D NoCs, in: Proceedings of the 14th IEEE Euromicro Conference on Digital System Design, DSD'11, IEEE, 2011, pp. 634–642.

Please cite this article in press as: M. Kamali, et al., Formal development of wireless sensor–actor networks, Science of Computer Programming (2012), doi:10.1016/j.scico.2012.03.002

# Paper II

## A Distributed Implementation of a Network Recovery Algorithm

**Maryam Kamali, Linas Laibinis, Luigia Petre and Kaisa Sere**

Originally published in: *International Journal of Critical Computer-Based Systems (IJCCBS)*, Vol. 4, No. 1, pp. 45-68. Inderscience Publishers, 2013.

# A distributed design of a network recovery algorithm

Maryam Kamali*, Linas Laibinis, Luigia Petre and Kaisa Sere

Department of Information Technologies,
Åbo Akademi University,
Joukahainenkatu 3–5 A, 20520 Turku, Finland
E-mail: maryam.kamali@abo.fi
E-mail: linas.laibinis@abo.fi
E-mail: luigia.petre@abo.fi
E-mail: kaisa.sere@abo.fi
*Corresponding author

**Abstract:** The increase in design complexity emphasises the relevance of formal verification techniques for both software and hardware. Formal methods with their mathematical-based modelling can provide proofs of various properties for the designs, thus ensuring a certain degree of complexity control and enhancing the system confidence. There are numerous formal modelling and verification techniques employed in designing complex systems. Typically, they either prove or disprove the correctness of the particular specifications of a system's algorithms with respect to certain initial requirements. The Event-B formal method has been recently extended to address the gap between specification and implementation, via the so-called modularisation extension. In this paper, we present a modularisation-based derivation of a distributed design for a network recovery algorithm, based on the refinement technique of Event-B. We thus contribute to enhancing the reliability and availability of network designs.

**Keywords:** wireless sensor-actor networks; WSANs; network recovery algorithm; distributed design; object-orientation; formal method; Event-B; refinement; modularisation; Rodin-tool.

**Reference** to this paper should be made as follows: Kamali, M., Laibinis, L., Petre, L. and Sere, K. (2013) 'A distributed design of a network recovery algorithm', *Int. J. Critical Computer-Based Systems*, Vol. 4, No. 1, pp.45–68.

**Biographical notes:** Maryam Kamali received her MSc in Information Technologies from the Iran University of Science and Technology since 2008. From 2009, she has been working in the Distributed Systems Laboratory of the Turku Centre for Computer Science (TUCS) Graduate School where she is a PhD student, enrolled at Åbo Akademi University. Her research interests include applying formal methods for modelling and verification of communication designs in wireless sensor network and network-on-chip.

Linas Laibinis is a Senior Researcher at Åbo Akademi University and holds a docent degree in Computer Science since 2011. He received his PhD in Computer Science in 2000 on mechanised formal reasoning about computer programmes. His research interests include interactive environments for proof

Copyright © 2013 Inderscience Enterprises Ltd.

and program construction, as well as application of formal methods to modelling and development of fault tolerant and/or communicating software systems. He has published about 60 refereed articles.

Luigia Petre is a Senior Lecturer at Åbo Akademi University and holds a docent degree (the equivalent of Habilitation) in Computer Science since 2012. She received her PhD in Computer Science in 2005 on modelling techniques in formal methods. Her research interests include formal methods and their integration, energy-aware computation, network availability, network-on-chip architectures, wireless sensor-actor networks, multi-core systems, time and space dependent computing and meta-modelling. She has about 40 publications.

Kaisa Sere is a Professor in Computer Science and Engineering at Åbo Akademi University since 1997. She received her PhD in 1990 on the formal design of parallel algorithms. She is the founder and leader of the Distributed Systems Laboratory at the Department of Information Technologies at the same university. Her current research interests are within the design of distributed systems, especially refinement-based approaches to the construction of systems ranging from pure software to hardware and digital circuits. She was the leader of the Åbo Akademi University work in the EU IST projects Matisse, Rodin and Deploy. She has supervised 17 PhD theses and currently (co-)supervising seven PhD students. She has published more than 100 refereed articles. She was the General Chair of the international conference Formal Methods 2008, the flagship conference in the field of formal methods.

This paper is a revised and expanded version of a paper entitled ‘Reconstructing coordination links in sensor-actor networks’ presented at Nodic Workshop on Dependability and Security (NODES), Copenhagen, Denmark, 22–23 April 2010.

## 1 Introduction

Our society relies on software-intensive systems or, differently formulated in Booch (2007) “software lives in the interstitial spaces of our society”. Networked systems are one example of the software omnipresence in the everyday life. The financial systems, including banking and stock exchanges, the travelling systems, including booking flights and hotels, the electric grids administration, the nuclear plants processes, etc., are all examples of essential components of the society being gradually adapted to functioning on-line, via the internet or other networking configurations.

Given the widespread and relevance of software-intensive systems, it is imperative to be able to rely on software, i.e., to be certain of its various features and properties. Formal methods, with their mathematical proving core, are an important instrument in ensuring the integrity of software-intensive systems. Traditionally characterised as hard

to use, due to the requested mathematical background and the lack of automatic tools, nowadays formal methods have matured, to the point where they are considered in industry when developing software-intensive systems (Woodcock et al., 2009). Examples of the industrial undertaking of formal methods are increasing. The famous line 14 of the driver-less Parisian metro (Gerhart et al., 1994), developed in 1998 using the B-method (Abrial, 1996), is the first notable example of a formal method-based development, reviewed in Lecomte (2009). The method used by Siemens for developing the software controlling the line 14 train ensured its correctness in a mathematical manner that effectively eliminated the unit testing from the software life-cycle. Few human resources are now needed to operate the trains and in addition, the trains are faster, hence fewer are needed in total.

More recent examples of the Event-B (Abrial, 2010) formal method usage in industry can be seen for instance with space systems (Fathabadi et al., 2011) and SAP (Jeremy and Wei, 2010). In Event-B, the development of a model is carried out step by step from an abstract to more concrete specifications. Using the *refinement* approach, a system can be described at different levels of abstraction, and the consistency in and between levels can be proved mathematically. An *ideal scenario* of formal methods supporting the development of reliable software-intensive systems consists in getting a formal proof that the final software is a correct implementation of the specification. For Event-B, this means *in practice* that there is a series of intermediate formal models between the specification and implementation, all proved correct together with their derivation from each other. We are now moving one step closer in Event-B towards the ideal scenario, in that we can verify that our derived design is correctly distributed. Consider the example of a network recovery algorithm, where some feature requires the addition of a link among two nodes. We typically specify this feature in Event-B and model its correctness and properties in a centralised manner, for the network as a whole. However, at the implementation phase, this feature has to involve the two network nodes that synchronise, so that each adds the required reference to the other. We need a methodology for transforming our centralised modelled feature of adding a link into a distributed addition of a link among the two nodes. Such a methodology has been recently proposed and is referred to as the *modularisation extension* (Iliasov et al., 2010) of Event-B.

In this paper, we start from a previously developed (Kamali et al., 2008) and verified (Kamali et al., 2010) recovery algorithm for wireless sensor-actor networks (WSANs) and present the formal derivation of a distributed design for this algorithm. This derivation is highly significant as a contribution to provide reliable *and* distributed software specifications. Our derived distributed design for the network recovery algorithm has a visible object-orientation style, thus bringing along two more features. First, it supports modifiability and reusability and second, it facilitates an easier translation to object-oriented code. Our work also addresses the availability research, because we formalise the distribution of a fault removal algorithm for *network recovery*.

Event-B (Abrial, 2001, 2007, 2010) is an extension of the B formalism (Abrial, 1996) for specifying distributed and reactive systems. A system model is gradually specified on several levels of abstraction, always ensuring that a more concrete model is a *correct implementation* of an abstract model. The language and proof theory of Event-B are based on logic and set theory. The correctness of the stepwise construction of formal models is ensured by discharging a set of proof obligations: if these obligations hold, then the development is mathematically shown to be correct. Event-B

comes with the associated tool Rodin (Abrial, 2007; Rodin Tool Platform, 2006), which automatically discharges part of the proof obligations and also provides the means for the user to discharge interactively the remaining proofs.

We proceed as follows. In Section 2 we shortly overview Event-B, the modularisation extension and the network recovery algorithm. In Section 3 we describe the development of the verified algorithm to the extent needed in this paper. In Section 4 we introduce and discuss our distributed design. We bring forward the impact of our modelling and its advantages in Section 5 and conclude in Section 6, also reviewing related work.

## 2 Preliminaries

In this section, we briefly overview Event-B, the modularisation extension, and the network recovery algorithm to distribute.

### *2.1 Event-B*

A typical Event-B model consists of two components, called *context* and *machine*. The context describes the static part of the model, i.e., it introduces carrier sets $s$ and constants $c$. The properties of these are described as a list of *axioms*. The machine describes the dynamic part of the model, consisting of a list of *variables* $v$ and a list of *events* $E$. The set of values of the variables forms the *state* of the model. The properties that should be preserved during the execution are formulated as a list of *invariant* predicates over the state of the model.

An event, modelling state changes, is formed of a *guard* $G$ and a *substitution* $S$; the latter describes the next-state relation between the variables values $v$, before the occurrence of the event and the variables values $v'$, after the occurrence of the event. Additionally, it may contain new local variables (parameters) $vl$. We illustrate the format of an event below:

$$E = \text{ANY } vl \text{ WHERE } G(c, s, vl, v) \text{ THEN } S(c, s, vl, v, v') \text{ END}$$

Event parameters and guards may be sometimes absent, in which case we respectively write 'WHEN' instead of 'ANY $vl$ WHERE' and 'BEGIN' instead of 'ANY $vl$ WHERE $G(c, s, vl, v)$ THEN'.

The guard $G$ is the necessary condition under which an event might occur; if the guard holds, we call the event *enabled*. The substitution $S$ is expressed as either a deterministic or a non-deterministic assignment to the variables. $S$ is often referred to as the *action* part of the event. The action determines the way in which the state variables change when the event occurs. For initialising the system, a sequence of actions is defined. When the guards of several events hold at the same time, then only one event is non-deterministically chosen for execution. If some events have no variables in common and are enabled at the same time, then they can be considered to be executed in parallel since their sequential execution in any order gives the same result.

A model is developed by a number of correctness preserving steps called *refinements*. One form of model refinement can add new data and new behaviour events on top of the already existing data and behaviour but in such a way that the

introduced behaviour does not contradict or take over the abstract machine behaviour. In this *superposition* refinement (Katz, 1993; Back and Sere, 1996), we present events in a refined model by using the shorthand notation "refined_event extends abstract_event". The meaning of this notation is that the refined event is created from the abstract one by simply adding new guards and/or new actions. Only the added elements are shown in the extended event, while the old guards and actions are implicitly present. In addition to the superposition refinement we may also use other refinement forms, such as *algorithmic refinement* (Back and Sere, 1989). In this case, an event of an abstract machine can be refined by several corresponding events in a refined machine. This will model different branches of execution, that can, for instance, take place in parallel and thus improve the algorithmic efficiency.

### *2.2 The modularisation extension*

Recently, the Event-B language and tool support have been extended with a possibility to define *modules* (Iliasov et al., 2010, 2011; Rodin Modularisation Plug-in, 2011) – i.e., components containing groups of callable atomic operations. Modules can have their own (external and internal) state and invariant properties. An important characteristic of modules is that they can be developed (refined) separately and, when needed, composed with the main system.

A module description consists of two parts – a *module interface* and a *module body*. Let $M$ be a module. A module interface $MI$ is a separate Event-B component. It allows the user of the module $M$ to invoke its operations and observe the external variables without having to inspect the module implementation details. $MI$ consists of external module variables $w$, constants $c$, sets $s$, the external module invariant $M_Inv(c, s, w)$, and a collection of module operations; the latter can have their own local variables $p$ and are characterised by their pre- and postconditions, as shown in Figure 1.

**Figure 1** Interface component

**Interface** $MI$
**Sees** $MI_Context$
**Variables** $w$
**Invariants** $M_Inv(c, s, w)$
**Initialisation** $\cdots$
**Process**
$PE_i =$ **ANY** $vl$ **WHERE** $g(c, s, vl, w)$ **THEN** $S(c, s, vl, w, w')$ **END**
$\ldots$
**Operations**
$O_i =$ **ANY** $p$ **PRE** $PRE(c, s, vl, w, p)$ **POST** $POST(c, s, vl, w, w', p)$ **END**
$\ldots$

In addition, a module interface description may contain a group of standard Event-B events under the *Process* clause. These events model the autonomous thread of control of the module, expressed in terms of their effect on the external module variables. In other words, the module process describes how the module's external variables may change between operation calls.

A formal module development starts with the design of an interface. Once an interface is defined, it cannot be altered afterwards. This ensures that a module body may be constructed independently from a model relying on the module interface. A module body is an Event-B machine. It implements the interface by providing a concrete behaviour for each of the interface operations. A set of additional proof obligations are generated to guarantee that each interface operation has a suitable implementation.

When the module $M$ is imported into another Event-B machine (specified by a special clause *USES*), the importing machine can invoke the operations of $M$ and read the external variables of $M$. To make a module specification generic, in $MI_Context$ we can define some constants and sets (types) as parameters. The properties over these sets and constants define the constraints to be verified when the module is instantiated. The concrete values or constraints needed for module instantiation are supplied in the *USES* clause of the importing machine.

Module instantiation allows us to create several instances of the same module which are distinctive namely using the clause $prefix$. Different instances of a module operate on disjoint state spaces. Via different instantiations of generic parameters the designers can easily accommodate the required variations when developing components with similar functionality. Hence module instantiation provides us with a powerful mechanism for reuse.

The latest developments of the modularisation extension also allows the developer to import a module with a given concrete set as its parameter. This parameter becomes the index set of module instances. In other words, for each value from the given set, the corresponding module instance is created. Since each module instance operates on a disjoint state space, parallel calls to operations of distinct instances are possible in the same event.

A general strategy of a distributed system development in Event-B is to start from an abstract centralised specification and incrementally augment it with design-specific details. When a suitable level of details is achieved, certain events of the specification are replaced by the calls of interface operations and variables are distributed across modules. As a result, a monolithic specification is decomposed into separate modules. Since decomposition is a special kind of refinement, such a model transformation is also correctness-preserving. Therefore, refinement allows us to efficiently cope with the complexity of distributed systems verification and gradually derive an implementation with the desired properties and behaviour.

### *2.3 The recovery algorithm in WSANs*

WSANs are a rather new generation of sensor networks (Akyildiz and Kasimoglu, 2004), made of two kinds of nodes: sensors and actors. In a WSAN, sensors detect the events that occur in the field, gather them and transmit the collected data to actors. The actors react to the events in the environment based on the received information. The sensor nodes are low-cost, low-power devices equipped with limited communication capabilities, while the actor nodes are usually mobile, more sophisticated and powerful devices compared to the sensor nodes. In addition, the density of sensor nodes in WSANs is much bigger than that of actor nodes. WSANs are dynamic networks where the network topology continuously changes because some new links or nodes are added, or are removed due to hardware crashes, lack of energy, malfunctions, etc.

WSANs have been applied in a variety of commercial, industrial and military applications to react to the situations sensed in the environment. For example, WSANs are installed in forests to prevent and/or resolve forest fires; the firefighter-actors are expected to stop the spread of the fire immediately. Therefore, in real-time applications of WSANs, a fast and effective response is a key concern that can only be provided with a reliable actor-actor coordination. The actor-actor coordination mechanisms provide means for actors to share information and take decisions on the proper reactions. In order to have real-time actor-actor coordination in WSANs, a self-reconfigurable, reliable and real-time communication approach is necessary. Different real-time communication protocols and a number of self-reconfigurable recovery mechanisms have been proposed in the literature (Imran et al., 2012; Gungor et al., 2008; Ngai et al., 2006). However, in most of the cases, the reliability of the developed recovery mechanisms is justified based on simulation results, that cannot guarantee the correctness of these mechanisms.

In this paper we describe a model of the basic functioning of a WSAN, focusing on the actors and their communication links with each others. Besides modelling basic network functioning, we model an abstract recovery algorithm for failed communication links, generated when an actor fails. The neighbours of a failed actor are able to reestablish communication links among themselves based only on localised knowledge of 1-hop (direct) neighbours and 2-hop neighbours (neighbours of neighbours). In the simulated algorithm (Kamali et al., 2008) as well as in the detailed formal model of the algorithm (Kamali et al., 2010), the reestablished links among the neighbours of a failed node are based on the underlying sensor network. However, in this paper we are focusing on the distributed nature of the recovery and do not go into the full details of the algorithm.

The formalisation of the algorithm for self-recovering actor coordination is proposed in Kamali et al. (2012) using Event-B and the associated Rodin platform. The formalism is used to verify essential properties such as the functional correctness and the termination of the recovery mechanism. An important aspect of the algorithm is that indirect links between actors are built in a distributed manner, thus ensuring the *self*-recovering of the network. To model the functional correctness of the recovery algorithm, authors use the mathematical concepts of tree and forest. In graph theory, a tree is a graph whose any two vertices are connected by a non-cyclic path, while a forest is a set of disjoint trees. A special data structure is introduced in Kamali et al. (2012) to model a forest and prove the correctness properties.

## 3 Formal specification of the network recovery algorithm

In this section, we present a model of the recovery algorithm in two abstraction steps.

### *3.1 The initial model*

In the context of our initial model we define the finite and non-empty set $NODE$ of all the network nodes. These nodes can be either sensor nodes or actor nodes, hence we partition the set $NODE$ into the subsets $sensors$ and $actors$. We also define the generic set $STATUS = \{ok, fail\}$, where the constant $fail$ denotes the failed status of

a node and the constant $ok$ denotes the non-failed status of a node. The initial status of the nodes is defined with the constant function $initial_status \in NODE \rightarrow STATUS$.

In the machine of our initial model we define four variables. The status of each node (non-failed or failed) is modelled with the function $Status \in NODE \rightarrow STATUS$. The relation $ANET \in actors \leftrightarrow actors$ denotes the bidirectional, non-failed actor links. This relation is irreflexive and symmetric. The $ANET$ relation stores only the direct links between actors. For our development purposes, we define the variable $recovered_ANET \in actors \leftrightarrow actors$ to store the direct links in $ANET$ together with the indirect links that are established by the recovery mechanism. The relation $recovered_ANET$ is irreflexive and symmetric as well. The set $FailedNodeNeigh \subseteq actors$ models non-failed actors, more precisely the actor neighbours of a failed actor. This set is updated when an actor is detected as failed, as shown shortly. We use the annotation *@acti* and *@grdj* to be able to refer to the different actions *i* and guard conditions *j* of an event.

The *INITIALISATION* event sets the status of all the nodes based on the $initial_status$ constant function. The $ANET$ and $recovered_ANET$ relations are set to be empty. The set $FailedNodeNeigh$ is set to $\varnothing$.

**INITIALISATION**
**BEGIN**
  **@act1** $Status := initial_status$
  **@act2** $ANET := \varnothing$
  **@act3** $recovered_ANET := \varnothing$
  **@act4** $FailedNodeNeigh := \varnothing$
**END**

Except the initialisation, the events in the initial model activate actor nodes from failed to non-failed (*ActivateNode*), add actor links (*AddLink*), deactivate actor nodes and remove their corresponding links (*DeactivateNode*), and abstractly recover connections when an actor fails (*Recovery1* and *Recovery2*). Thus, starting from an arbitrary, non-deterministic set of failed and non-failed actors that have no links with each other, as established by the *INITIALISATION* event, we can randomly activate actors, add links between non-failed actors, and deactivate actors and remove their corresponding links. The latter event models actor failures and enables our recovery mechanism. Until the recovery is complete, the first three events (*ActivateNode*, *AddLink*, and *DeactivateNode*) are not enabled anymore. We thus have a normal operation phase of the algorithm, when the *ActivateNode*, *AddLink*, and *DeactivateNode* events are non-deterministically executed and a recovery phase of the algorithm, when only *Recovery1* and *Recovery2* events are executed. The phase separation is modelled using the $FailedNodeNeigh$ variable. While $FailedNodeNeigh = \varnothing$, the algorithm is in its operational phase. While $FailedNodeNeigh \neq \varnothing$, the algorithm is in its recovery phase. A separate event *Recovery3* is defined to complete the recovery phase.

In the *ActivateNode* event, the status of a failed (*@grd2*) actor (*@grd1*) is changed to non-failed (*@act1*). The event is enabled if $FailedNodeNeigh = \varnothing$ (*@grd3*), as this is an operational event.

**ActivateNode**
**ANY** n **WHERE**
  **@grd1** $n \in actors$
  **@grd2** $Status(n) = fail$
  **@grd3** $FailedNodeNeigh = \varnothing$
**THEN**
  **@act1** $Status(n) := ok$
**END**

In the *AddLink* event we add a link between two distinct, non-failed actors (*@grd1-3*) that are not connected (*@grd4*) and are within communication range of each other (*@grd5*). Here, $r_a$ is the communication range among two actors and $dist(u \mapsto v)$ denotes the Euclidean distance between the actors $u$ and $v$. Namely, if $u_x$ and $u_y$ are the horizontal and vertical Cartesian coordinates of the node $u$, respectively, and $v_x$ and $v_y$ are the horizontal and vertical Cartesian coordinates of the node $v$, respectively, then $dist(u, v) = \sqrt{(u_x - v_x)^2 + (u_y - v_y)^2}$. As $ANET$ is symmetric, we add the link in both directions in $ANET$ (*@act1*) and remove it (in both directions) from $recovered_ANET$. This corresponds to cancelling the (potential) temporary links proposed by the recovery algorithm. When two nodes can be connected directly, an indirect link consuming more power is not needed. Therefore, the constructed indirect links are removed from $recovered_ANET$. The event is enabled only if $FailedNodeNeigh = \varnothing$ (*@grd6*), as this is an operational event.

**AddLink**
**ANY** n m **WHERE**
  **@grd1** $n \in actors \wedge m \in actors$
  **@grd2** $Status(n) = ok \wedge Status(m) = ok$
  **@grd3** $n \neq m$
  **@grd4** $n \mapsto m \notin ANET$
  **@grd5** $dist(n \mapsto m) < r_a$
  **@grd6** $FailedNodeNeigh = \varnothing$
**THEN**
  **@act1** $ANET := ANET \cup \{n \mapsto m, m \mapsto n\}$
  **@act2** $recovered_ANET := recovered_ANET \setminus \{n \mapsto m, m \mapsto n\}$
**END**

The *DeactivateNode* event changes the status of a non-failed actor (*@grd1-2*) to that of a failed one (*@act1*); also, all the links of that actor are removed from both $ANET$ and $recovered_ANET$, expressed with the domain subtraction operator $\lhd$ and the range subtraction operator $\rhd$ (*@act2-3*). The domain and range subtraction operators on $ANET$ and $recovered_ANET$ relations denote only those pairs whose first and second element is not $n$. In addition, the neighbours of the failed actor become members of the $FailedNodeNeigh$ set (*@act4*).

The *DeactivateNode* event is an operational one, enabled only when $FailedNodeNeigh = \varnothing$ (*@grd3*). If the failing actor had neighbours in $ANET$, then $FailedNodeNeigh \neq \varnothing$ after *@act4* is executed. At this point, the algorithm has entered into the recovery phase. In our algorithm, we model the situation where more than one actor can fail at a time only if these actors have no neighbours in $ANET$. If a node with neighbours fails, then no othr node can fail again until we have recovered the communication between the neighbours of the failed node. This is modelled by

the *DeactivateNode* event not being enabled again until $FailedNodeNeigh$ becomes empty again (*@grd3*).

**DeactivateNode**
**ANY** $n$ **WHERE**
 **@grd1** $n \in actors$
 **@grd2** $Status(n) = ok$
 **@grd3** $FailedNodeNeigh = \varnothing$
**THEN**
 **@act1** $Status(n) := fail$
 **@act2** $ANET := \{n\} \lhd\!\!\!- ANET \,-\!\!\!\rhd \{n\}$
 **@act3** $recovered_ANET := \{n\} \lhd\!\!\!- recovered_ANET \,-\!\!\!\rhd \{n\}$
 **@act4** $FailedNodeNeigh := ANET[\{n\}]$
**END**

When an actor with neighbours has been deactivated, we need to check whether a recovery is needed. The event *Recovery1* is enabled when two neighbours of a deactivated actor (*@grd1*) have no short connection through other neighbours (*@grd2*) (i.e., there is no path from one actor to the other at most 2-hop long). We notice that this check does not imply that the deactivation of an actor has led to partitioning the actor network, although in some cases, it may have led. If *Recovery1* is enabled, then a direct actor-actor link is established and stored separately from $ANET$, in $recovered_ANET$ (*@act1*). This separation is essential because $ANET$ models only the real direct links among nodes. The recovered links modelled by $recovered_ANET$ are not direct links, i.e., nodes are not within communication range of each other; they can typically only communicate through some intermediate nodes which are not modelled at this abstract level. As $FailedNodeNeigh$ is a subset of the finite set $NODE$, we observe that the *Recovery1* event can be enabled only a finite number of times, hence the recovery phase terminates. Technically, this is true because $card(FailedNodeNeigh)$ decreases at each execution of *Recovery1* and eventually the guard of *Recovery1* will hold no longer. We observe that node $n$ cannot be equal to node $k$ due to *@grd2* ($ANET; ANET$ is reflexive). $ANET; ANET$ denotes the forward relational composition of $ANET$ by itself.

**Recovery1**
**ANY** n k **WHERE**
 **@grd1** $n \in FailedNodeNeigh \land k \in FailedNodeNeigh \land n \neq k$
 **@grd2** $n \mapsto k \notin ANET \land n \mapsto k \notin ANET; ANET$
**THEN**
 **@act1** $recovered_ANET := recovered_ANET \cup \{n \mapsto k, k \mapsto n\}$
 **@act2** $FailedNodeNeigh := FailedNodeNeigh \setminus \{n\}$
**END**

**Recovery2**
**ANY** n k **WHERE**
 **@grd1** $n \in FailedNodeNeigh \land k \in FailedNodeNeigh$
 **@grd2** $n \mapsto k \in ANET \lor n \mapsto k \in (ANET; ANET) \setminus (actors \lhd id)$
**THEN**
 **@act1** $FailedNodeNeigh := FailedNodeNeigh \setminus \{n\}$
**END**

**Recovery3**
**WHERE**
**@grd1** $card(FailedNodeNeigh) = 1$
**THEN**
**@act1** $FailedNodeNeigh := \varnothing$
**END**

The *Recovery2* event treats the situation when a failure is detected but an alternative path through 1-hop or 2-hop neighbours already exists between the neighbours of the deactivated actor ($n \mapsto k \in ANET \vee n \mapsto k \in ANET; ANET$). In this case, $FailedNodeNeigh$ is simply updated. In *Recovery2* we are also sure that $n$ and $k$ cannot be equal (*@grd2*). The last element of $FailedNodeNeigh$ is removed via the *Recovery3* event.

Overall, in this model we have described the non-deterministic activation and deactivation of actor nodes and the adding and removing of actor links in a dynamic (wireless sensor-actor) network for whom a communication problem among the actors can be detected and recovered from via direct actor links. The recovery algorithm assumes some global network knowledge for the recovery, expressed by $ANET; ANET$. Also, the recovery mechanism establishes direct links among the non-failed actor neighbours of the failed actor. We observe that both the recovery assumption and the recovery mechanism can be used in practice only for *strategic actors*, i.e., actors whose range is sufficiently large to check the contents of $ANET; ANET$ and establish direct actor links. The following model considers more localised assumptions as well as indirect recovery paths.

### *3.2 The refined model*

In the initial model we have defined the actor network recovery based on global knowledge about the whole network ($ANET; ANET$). In this model we assume that each actor has access only to information about its 1-hop neighbours and 2-hop neighbours, i.e, we restrict the actor-actor communication recovery to take place via local information. This model is a refinement of the initial model. The goal of this refinement step is to supplement the global knowledge of the network in the initial model ($ANET; ANET$) with localised knowledge for every actor. We achieve this by introducing a new variable, the irreflexive relation $l_net \in (actors \times actors) \leftrightarrow NODE$ that, for each actor, keeps track of the 1-hop and 2-hop neighbours and the recovered links modelled by $recovered_ANET$ in the initial model. The relation $l_net$ shows a specific distributed actor-actor communication model. The meaning of this relation is that a 1-hop neighbour $m$ of an actor $n$ is denoted by $n \mapsto m \mapsto m \in l_net$ and a 2-hop neighbour $m$ of an actor $n$ is denoted by $n \mapsto m \mapsto k \in l_net$. In the first example, $m$ is locally related to $n$ via $m$ (itself, i.e., via a direct link) and in the second example $m$ is locally related to $n$ via $k$ (i.e., $m$ is a 2-hop neighbour of $n$, while $k$ is a 1-hop neighbour of $n$). The relation $l_net$ describes all these *localised* links between nodes. Those 2-hop links that are reestablished via nodes other than actors denote the $recovered_ANET$ links of the initial model.

When a new link is added between two actors, the $l_net$ relation also needs to be updated. Therefore, the *AddLink* event is extended to also add links to $l_net$. For every two actors $n$ and $m$ that have a direct link, $n \mapsto m \mapsto m$ and $m \mapsto n \mapsto n$ are added in

$l_net$ (*@act3*), meaning that $n$ has a link with $m$ through $m$ ($m$ is a 1-hop neighbour $n$) and $m$ has a link with $n$ through $n$ ($n$ is a 1-hop neighbour of $m$). Moreover, the indirect links between $n$ and $m$ through sensors (denoted by expression such as $\{m\} \times \{n\} \times (Status^{-1}[\{ok\}] \setminus actors)$ are removed from $l_net$ because there is now a direct link with less power consumption. This is similar to *@act2* of this event in the initial model.

**AddLink**
**extends** AddLink
**THEN**
**@act3** $l_net := l_net \cup \{n \mapsto m \mapsto m, m \mapsto n \mapsto n\} \setminus ((\{n\} \times \{m\} \times Status^{-1}[\{ok\}] \setminus actors) \cup (\{m\} \times \{n\} \times Status^{-1}[\{ok\}] \setminus actors))$
**END**

**Addl_net2hopLink**
**ANY** n m k **WHERE**
**@grd1** $Status(n) = ok \wedge Status(m) = ok \wedge Status(k) = ok$
**@grd2** $m \mapsto k \mapsto k \in l_net \wedge n \mapsto m \mapsto m \in l_net \wedge n \mapsto k \mapsto m \notin l_net \wedge k \mapsto n \mapsto m \notin l_net$
**@grd3** $m \neq n \wedge n \neq k$
**@grd4** $FailedNodeNeigh = \varnothing$
**THEN**
**@act1** $l_net := l_net \cup \{n \mapsto k \mapsto m, k \mapsto n \mapsto m\} \setminus ((\{n\} \times \{k\} \times Status^{-1}[\{ok\}] \setminus actors) \cup (\{k\} \times \{n\} \times Status^{-1}[\{ok\}] \setminus actors))$
**END**

The *Addl_net2hopLink* event is a newly introduced event that handles the addition of 2-hop neighbour links for actors. If an actor has a direct link with two other actors, then these actors will be 2-hop neighbours of each other. The actors involved in this event have to be non-failed (*@grd1*), have a direct (1-hop) link but not yet a 2-hop link through actors in $l_net$ (*@grd2*) and be distinct (*@grd3*). Also, this new event belongs to the operational phase of the algorithm, hence, $FailedNodeNeigh = \varnothing$ is part of its guard (*@grd4*). If these conditions are satisfied, then any 2-hop links between $n$ and $k$ reestablished through sensors are removed from $l_net$ and the detected 2-hop links through actors are added in $l_net$ in both directions (*@act1*).

When deactivating an actor node, all its links are also removed. Thus, in the *DeactivateNode* event the new action *@act4* removes all the links to and from the failed actor in the $l_net$ relation. The expression $\{n\} \times dom(ANET) \times Status^{-1}[\{ok\}]$ describes all the links from $n$, either direct connections (1-hop neighbours) or indirect connections (2-hop neighbours) and the expression $dom(ANET) \times \{n\} \times \{n\}$ describes all the direct connections (1-hop neighbours) links to $n$.

**DeactivateNode**
**extends** DeactivateNode
**THEN**
**@act4** $l_net := l_net \setminus ((\{n\} \times dom(ANET) \times Status^{-1}[\{ok\}]) \cup (dom(ANET) \times \{n\} \times \{n\}))$
**END**

We now need to model the detection of failed actors and the recovery of links based on local information instead of being based on the global actor-actor coordination as described by $ANET; ANET$. We now use $l_net$ information in addition to $ANET$ for detecting an actor failure (*@grd3*) and recovering links in the *Recovery1* event.

**Recovery1**
**extends** Recovery1
**ANY** n m k **WHERE**
**@grd3** $n \mapsto k \mapsto m \in l_net \wedge k \mapsto n \mapsto m \in l_net \wedge n \mapsto m \mapsto m \notin l_net \wedge$
$k \mapsto m \mapsto m \notin l_net \wedge n \mapsto k \mapsto k \notin l_net \wedge k \mapsto n \mapsto n \notin l_net$
**THEN**
**@act3** $l_net :| \; l_net' \subseteq (l_net \setminus \{n \mapsto k \mapsto m, k \mapsto n \mapsto m\}) \cup$
$(\{k\} \times \{n\} \times Status^{-1}[\{ok\}]) \cup (\{n\} \times \{k\} \times Status^{-1}[\{ok\}])$
$\cup \; indir(k, n) \cup indir{\sim}(n, k) \cup indir(n, k) \cup indir{\sim}(k, n)$
**END**

**Recovery2**
**extends** Recovery2
**ANY** n m k **WHERE**
**@grd3** $(n \mapsto k \mapsto m \in l_net \wedge n \mapsto m \mapsto m \notin l_net \wedge k \mapsto n \mapsto m \in l_net$
$\wedge k \mapsto m \mapsto m \notin l_net) \vee (n \mapsto k \mapsto k \in l_net \wedge k \mapsto n \mapsto n \in l_net)$
**THEN**
**@act2** $l_net := l_net \setminus (\{n \mapsto k \mapsto m, k \mapsto n \mapsto m\} \cup indir(n, m) \cup indir(k, m)$
**END**

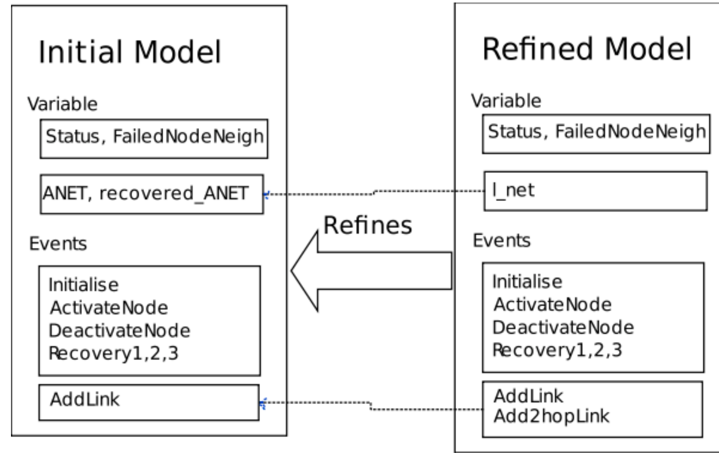
When actor $m$ is detected as failed, neighbours of $m$ ($n$ and $k$) that have a connection with each other through $m$ ($n \mapsto k \mapsto m$, $k \mapsto n \mapsto m \in l_net$) need to find an alternative path towards each other. If there is no other route in $n \mapsto m \mapsto m$, $k \mapsto m \mapsto m$,$n \mapsto k \mapsto k$ and $k \mapsto n \mapsto n \notin l_net$, then $l_net$ should be updated by removing invalid links and adding new routes.

Since $m$ is failed, links between $n$ and $k$ through $m$ are not anymore valid, so $n \mapsto k \mapsto m$ and $k \mapsto n \mapsto m$ are removed from $l_net$. Then we need to add new links to connect $n$ and $k$. In this refinement, we model that actor $n$ can establish a link with actor $k$ through any non-failed node: $\{n\} \times \{k\} \times Status^{-1}[\{ok\}]$ and similarly for actor $k$ to establish a new link with actor $n$: $\{k\} \times \{n\} \times Status^{-1}[\{ok\}]$. For clarity, we use the following short definitions: $indir(k, n) = ANET[k] \times n \times k$ and $indir{\sim}(n, k) = n \times ANET[k] \times k$. When node $n$ establishes a link with $k$, neighbours of $n$ also need to add node $k$ to their 2-hop neighbours list ($indir(n, k)$ and $indir{\sim}(k, n)$). Moreover, neighbours of $k$ need to add $n$ to their 2-hop neighbours list ($indir(k, n)$ and $indir{\sim}(n, k)$). The updating process of $l_net$ is described by *@act3* in the *Recovery1* event.

We also add a new action to the event *Recovery2* that updates $l_net$ by removing all the links to the failed actor or through it.

The relation $l_net$ is an elegant and very abstract data structure relating two actor nodes in its domain via a third node in its range. Indirect links between actors are now established non-deterministically based on localised information. The graphical overview of our model refinement is shown in Figure 2, where 'Initial Model' denotes the model described in Section 3.1 and 'refined model' denotes the model described in the current Section 3.2.

**Figure 2** Overview of our model refinement




## 4 Distributing the recovery algorithm

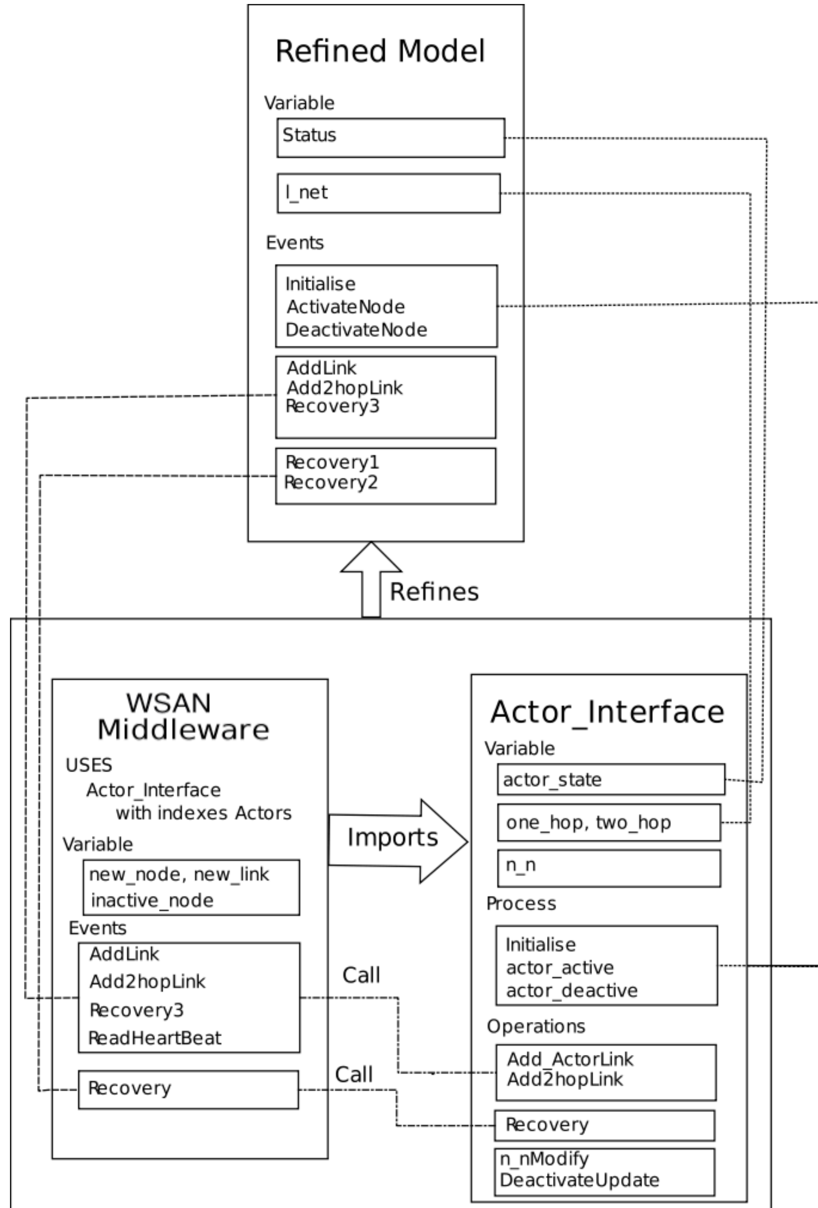
In this section we put forward the decomposition of our centralised WSAN model into a model for a WSAN infrastructure and a model for the distributed actor node(s). The correctness of this decomposition step is addressed as a special kind of model refinement.

### *4.1 A distributed model of the actor network*

In the initial model we capture the recovery algorithm in a centralised way. This centralisation is achieved by allowing the actor nodes to access global network links, modelled by $ANET$. However, in the following refinement step, the variable $l_net$ is introduced to limit the actor access to the local network links rather than the global ones. Such a refinement step allows us to express the distributed nature of the recovery algorithm in WSANs. Specifically, the refined model represents the communicating infrastructure *together with* actions of separate actors, so that everything is modelled in a single machine. On one hand, the modelling and the verification of properties in the network is simpler but, on the other hand, it causes difficulties for its *distributed implementation*. Therefore, in order to implement a distributed actor network, we need to separate the WSAN infrastructure and the individual actor operations.

The overall scheme of the decomposition refinement is shown in Figure 3. The model presented in Section 3.2 (‘refined model’ in Figure 3) is now split into two parts: an *interface* model, named ‘Actor_Interface’ and a *machine* model, named ‘WSAN Middleware’. The interface specifies the external state and the behaviour model of actor nodes, while the machine models the WSAN infrastructure that enables the nodes to communicate with each other. The actor interface is imported into the machine as a module, which is indexed by the set of actor nodes. This machine acts as the communication middleware providing access to module instances via the given interface.

**Figure 3** The structure of the decomposition refinement




## *4.2 Actor interface*

The actor interface defines several external variables that can be accessed for reading by the middleware. These variables model the visible state of an actor node. Each actor interface has four variables: $actor_state, one_hop, two_hop$ and $n_n$. The $actor_state$ variable represents the state of a node, which can be either *ok* or *fail*. This variable refines the corresponding variable $Status$ of the abstract model. The $one_hop$

and $two_hop$ variables store the direct and indirect neighbours of each node, thus corresponding to $ANET$ and $l_net$. The $n_n$ variable is a control variable that informs the WSAN middleware about recent state changes in an actor. If the value of $n_n$ is $TRUE$, this means that the node has recently became active and thus the link discovery should be started. Otherwise, the node is either inactive or the link discovery process has already been done.

$actor_state \in STATUS$
$one_hop \subseteq actors$
$two_hop \in actors \leftrightarrow actors$
$n_n \in BOOL$

**PROCESS**
$actor_active$
**WHEN**
$actor_state = fail$
**THEN**
$actor_state := ok$
$n_n := TRUE$
**END**

**PROCESS**
$actor_inactive$
**WHEN**
$actor_state = ok$
**THEN**
$actor_state := fail$
$n_n := FALSE$
**END**

**OPERATION**
$Add_ActorLink$
**ANY**
$new_node$
**PRE**
$actor_state = ok$
$new_node \notin one_hop$
**RETURN**
$r$
**POST**
$one_hop' = one_hop \cup \{new_node\}$
$two_hop' = two_hop \setminus \{new_node\} \times (NODE \setminus actors)$
$r' = one_hop$
**END**

**OPERATION**
$Add2hopLink$
**ANY**
$neigh$
**PRE**
$actor_state = ok$
$neigh \in actor \leftrightarrow actor$
**POST**
$two_hop' = two_hop \cup neigh$
**END**

Each actor can have its own autonomous process which is not dependent on the rest of the network. The events describing such a process are given in the *PROCESS* part of the actor interface. In our model, we have two events, $actor_active$ and $actor_deactive$, that change the state of a node from inactive to active and from active to inactive, respectively. In addition, the control variable is set to inform the communication middleware about the recent change in the node state.

In the operation part of the actor interface, the operations that can be called by the middleware are defined. The middleware calls these operations in its events to add direct and indirect links between nodes as well as recover from disconnectivity. In our interface, we have six operations where can be called by the communication middleware. However, only the three operations referring to the distributed aspect of the network recovery algorithm are discussed in this paper, for simplicity. We mention some of the other operations briefly when called in the events of the middleware. The

$Add_ActorLink$ adds a new link to a given node into the variable $one_hop$. The node is passed as a parameter of the operation call. Moreover, the call removes all the indirect links via sensors for the given node from the variable $two_hop$. This is because a direct actor link removes the need for indirect ones.

The $Add2hopLink$ operation updates the $two_hop$ variable by adding a new link, passed as the operation parameter and returns the *void* value to the middleware.

**OPERATION**
$Recovery$
**ANY**
$m \quad fn$
**PRE**
$actor_state = ok \quad \wedge \quad m \in one_hop$
**POST**
$one_hop' = one_hop \setminus \{m\}$
$(fn \notin one_hop \wedge fn \notin dom(two_hop \rhd \{m\}) \Rightarrow$
$two_hop' \subseteq (two_hop \rhd \{m\}) \cup \{fn\} \times (NODE \setminus actor))$
$(fn \in one_hop \vee fn \in dom(two_hop \rhd \{m\}) \Rightarrow two_hop' = two_hop \rhd \{m\})$
**END**

The $Recovery$ operation removes invalid links and adds a new indirect link, if needed. The failed node $m$ which is passed as an argument to the $Recovery$ operation is removed from the $one_hop$ list. Moreover, a neighbour $fn$ (a node in the $FailedNodeNeigh$ set in the previous models) of the failed node $m$ is passed to the $Recovery$ operation in order to reestablish a link through sensors, between the actor and $fn$. In the $Recovery$ operation we then check the $one_hop$ and $two_hop$ lists and if there is a link between the actor and $fn$, $two_hop$ list is just updated by removing all indirect links through the failed node. If there is a link neither in $one_hop$ list nor $two_hop$ list, $two_hop$ list is updated by removing all indirect links through the failed node $m$ and adding an indirect link to $fn$ through sensor nodes.

### *4.3 WSAN middleware*

In this section, we show how the defined interface can be used in the refined machine. Moreover, we discuss how the refined machine can act as a WSAN middleware.

**USES**
$actor_interface$ **with prefix** $actor$
**PROCESS LINK**
$actor_active : ActivateNode$
$actor_inactive : DeactivateNode$

In the machine part of the refined model, we import the actor interface indexed by the $actor$ set. As explained in Section *2.2*, in this way we introduce a number of module instances, each for a particular actor node in the network. In order to refine the abstract events of the model before decomposition, in some cases we need to link these events to the events that are now distributed among the module instances. Specifically,

$ActivateNode$ and $DeactivateNode$ are the abstract events that are linked to the corresponding events in the process part of the actor interface.

One of the goals of the decomposition refinement step is data refinement of the data structures modelling the actor coordination and recovery mechanism in WSANs by the ones representing the coordination model of individual actors. Therefore, we need to show that the $one_hop$ and $two_hop$ variables of the actor module instances are correct replacements (i.e., data refinements) of the $l_net$ variable from the abstract model. In addition, the abstract $Status$ variable is now data refined by $actor_state$ residing in module instances. The gluing invariants between abstract and concrete variables are as follows:

$$l_net = (id \lhd (one_hop \times one_hop)) \cup two_hop$$
$$status = actor_state$$

The remaining events of the refined machine can be split into two groups. The first group consists of a single (new) event describing how the middleware monitors the actor nodes, specifically looking for new activated nodes or previously active and currently failed nodes. The second group consists of several events specifying the middleware reactions in response to the detected changes. The corresponding reactions are specified using operation calls to the affected node instances. These events are refinements of the abstract events for adding new links and initiating the recovery mechanism.

**EVENTS**
  $ReadHeartBeat$
  **ANY**
    $n \quad new \quad fail$
  **WHERE**
    $n \in actors$
    $(n_n(n) = TRUE \wedge new_node = \varnothing \wedge new = n) \vee (new = \varnothing)$
    $(actor_state(n) = fail \wedge one_hop(n) \neq \varnothing \wedge inactive_node = \varnothing \wedge fail = n)$
    $\vee(fail = \varnothing)$
  **THEN**
    $new_node := new$
    $inactive_node := fail$
    $FailedNodeNeigh := DeactivateUpdate(fail)$
  **END**

The new monitoring event is called $ReadHeartBeat$. As node activation and deactivation independently happens in a node and the network is not immediately aware of these changes, this event alarms the communication middleware for any changes in the network topology. The detected changes are stored into two new variables $new_node$ and $inactive_node$, which represent the indexes of, respectively, a node that was recently activated in the network and a node that recently failed (i.e., became inactive). In its guard, the $ReadHeartBeat$ event checks the state of actor node instances. If the $n_n$ value of the actor node is $TRUE$ then $new_node$ is updated to the actor index. Moreover, if the $actor_state$ value of the actor is *fail* and *one_hop* is not empty, i.e., the actor node has recently failed and recovery should be started, then the $inactive_node$ variable is also updated by the actor index. Finally, the $DeactivateUpdate$ operation of the failed node is called to set the list of $one - hop$ and $two - hop$ with $\varnothing$ and return the neighbours of the node to update $FailedNodeNeigh$. When $one - hop$ list

of a failed node is set to $\varnothing$ denotes that the node failure has been considered and the recovery links had been reestablished after detection of the failed node. Guards of the $ReadHeartBeat$ event corresponds to conditions that if a node has recently added or removed from the network which are modelled by evaluating the value of $n_n$ and $one_hop$ in each node, repectively.

The second group of events consists of the events $AddLink$, $Add2hopLink$, and $Recovery$. The $AddLink$ event is enabled when a new node joins the network and an active actor is in its communication range. By calling the $Actor_AddLink$ operations of these two neighbours, the corresponding internal lists of 1-hop neighbours of these nodes are updated, while the lists of 2-hop neighbours are returned to the $AddLink$ event. The returned lists are saved into two variables $m_neigh$ and $n_neigh$ to be used in the $Add2hopLink$ event to update the lists of 2-hop neighbours. Note the syntax of operation calls including two pairs of parentheses: in the first parentheses, the index of a called node instance is given, while, within the second parentheses, the concrete arguments of the procedure call are passed to the given module instance.

In the $Add2hopLink$ event, the $Add2hopLink$ operation is called for different module instances and their results are returned to the event. Since the return value of the $Add2hopLink$ operation is *void*, we have to assign it to the corresponding variables of the type $VOID$. However, for the sake of clarity, we introduce a shorthand notation for such calls: instead of $void_var := Operation(index)\ (parameters)$ we simply write $Operation\ (index)\ (parameters)$. Please also note that the module instance index values used within operation calls in this event are actually not single values but sets of indexes. In such a way, we can specify a multiple (broadcasting) call, when the same operations of a number of affected module instances are called simultaneously. Besides, the $n_nModify$ operation updates the value of $n_n$ from $TRUE$ to $FALSE$ and is called for $new_node$. The $n_n$ variable of $new_node$ is updated to denote that the $one_hop$ and $two_hop$ lists of $new_node$ and its neighbours are updated. Updating of $n_n$ variable in a node causes the communication middleware to not consider the node as a new node anymore.

**EVENTS**
$AddLink$
**ANY**
$n \quad m$
**WHERE**
$n \in new_node$
$m \in actors \land n \neq m$
$dist(n \mapsto m) < r_a$
**THEN**
$m_neigh := (Actor_AddLink(m)(n)) \times \{m\}$
$n_neigh := (Actor_AddLink(n)(m)) \times \{n\}$
$new_link := \{n, m\}$
**END**

**EVENTS**
$Add2hopLink$
**ANY**
$n \quad m$
**WHERE**
$n \in new_node \land m \in new_link \land n \neq m$
**THEN**
$Add2hopLink(dom(n_neigh))(\{m \mapsto n\})$
$Add2hopLink(dom(m_neigh))(\{n \mapsto m\})$
$Add2hopLink(dom(n))(m_neigh)$
$Add2hopLink(dom(m))(n_neigh)$
$new_link := \varnothing$
$new_node := \varnothing$
$n_nModify(new_node)$
**END**

Finally, the $Recovery$ event is enabled when a node failure is detected by the $ReadHeartBeat$ event. Since the recovery mechanism is now distributed among the actor nodes, in the refined machine we only call the recovery operation of neighbours of a failed node. Therefore, we merge the $Recovery1$ and $Recovery2$ events of the abstract machine into the $Recovery$ event of the refined model.

**EVENTS**
  $Recovery$
  **refines**
    $Recovery1 \quad Recovery2$
  **ANY**
    $n \quad k$
  **WHERE**
    $n \in FailedNodeNeigh$
    $k \in FailedNodeNeigh$
  **THEN**
    $Recovery(n)(inactiveNode \mapsto k)$
    $Recovery(k)(inactiveNode \mapsto m)$
    $FailedNodeNeigh := FailedNodeNeigh \setminus \{n\}$
  **END**

## 5 Contribution

In this paper we have employed Event-B to uncover a distributed software design for a network recovery algorithm. This algorithm has been formalised before in Event-B (Kamali et al., 2010) and its correctness and termination properties proved based on the Rodin platform (Kamali et al., 2012). The algorithm deals with recovering communication links between actors when intermediary actors fail. The nature of WSANs imposes that every useful recovery algorithm be distributed, so that it is taken care of by individual actors, based on some given infrastructure. In the previous works on this algorithm (Kamali et al., 2010, 2012), the actor actions and the infrastructure are all integrated in one specification. This obviously simplifies the understanding of the involved mechanisms as well as the formulation and proofs of the properties. However, it is problematic to implement such a specification. We address this difficulty in this paper and propose a distributed design for the network recovery algorithm. As we carry out our derivation still in Event-B, we can keep all the enumerated advantages of simplicity and proving. In addition, we get a distributed design, much easier to implement.

Our approach for distribution follows the object-oriented paradigm. Since there is a number of entities of the same kind in the network, i.e., network nodes, a class (indexed module) of nodes can be developed separately by modelling their local variables and behaviours. Specifically, following our modelling methodology, we introduce a module interface that models the state and the behaviour of individual nodes. In such a way, we map the notion of a class in object-oriented programming to the notion of of an indexed module in Event-B. Similarly to creating new objects of a class in object-oriented programming, in our model we import an interface with an index set as its parameter. Such an approach lets us distribute models of individual nodes in the network so that they can be more transparently transformed into a part of distributed network

implementation. The decomposed model provides enough intuition towards creating object-oriented code out of it.

Another interesting aspect of our derived distributed design refers to the failing actors. Actors can fail and be activated non-deterministically both in the integrated specification and in the distributed one. In the integrated specification however, only one node could fail at a time and the algorithm takes care of recovering its neighbours communication. Only after that can the normal operation of the actor, including other possible actor failures, occur. In the distributed design, any number of actors can fail simultaneously or sequentially. The distributed recovery algorithm takes care of recovering the communication of each failed actor's neighbours, in a non-deterministic order and (still) one at a time. This is a significant reduction of the previous constraints and constitutes an important contribution to modelling a reliable and more realistic distributed recovery.

## 6 Related work and conclusions

Several formal developments of complex systems have used the Event-B formal method, for instance network protocols (Abrial et al., 2003; Rehm, 2010), network-on-chip modelling (Kamali et al., 2011a, 2011b) and sensor networks (Kamali et al., 2010). Cansell and Méry (2006) model a development of a distributed reference counting algorithm by using the refinement technique. Hoang et al. (2009) develop a topology discovery algorithm in Event-B, to prove safety and liveness properties. The major contribution in these formal models consists in developing correct-by-construction models with the help of the refinement technique. Our contribution goes one step further, to decompose a verified development towards a distributed implementation.

Ball and Butler (2006) present a practical approach to the formal development of multi-agent systems in Event-B by using abstraction and decomposition techniques. The main purpose of using the decomposition method for the development is to cope with the complexity of the system. When the model is refined, it becomes more concrete and complex. Therefore, decomposition is ideal for simplifying the complexity of the model. In comparison, our aim is to apply the decomposition technique to approach an implementation of a distributed system.

Iliasov et al. (2011) decompose an integrated correct-by-construction specification of a distributed system to gain its distributed program. The paper addresses a state-based formal approach to correct-by-construction development of distributed programs. The authors present their approach by developing of a distributed leader election protocol as a case study. Decomposition refinement in the case study is accomplished by two instances of the module interface while in this paper we use indexed instances and approach an object-oriented implementation.

Ray and Cleaveland (2004) present a formal modelling of middleware-based distributed systems. They extend architectural integration diagrams (AIDs) (Ray and Cleaveland, 2003) by providing an operational semantics that supports formal analysis of distributed system designs. AID provides a mechanism that supports a wide variety of interprocess communication and frees the designers from manually modelling the different interprocess communications. Another middleware-based systems formalism is presented in Dan and Danning (2010). They work on semi-formal UML-based analysis using UML profiles and a notion of behavioural semantics. The main purpose of these

studies is the proposal of an operational semantics for modelling middleware-based distributed systems to analyse different designs. In this paper, we understand the middleware as a network infrastructure stepwise derived to achieve distribution.

The derivation of a distributed design from an integrated algorithm is very similar in nature to well-established research in formal methods, for instance in Back and Sere (1996). The derivation of a distributed broadcast algorithm from a centralised one is employed there as a case study and the formal development is carried out in action systems (Back and Kurki-Suonio, 1983) instead of Event-B. It is worth mentioning that actions systems are in fact a precursor of Event-B. However, Event-B has the associated Rodin platform, which makes proving an automated matter and thus brings a significant advantage to the establishment of reliable practices in software development.

## References

Abrial, J-R., Cansell, D. and Méry, D. (2003) 'A mechanically proved and incremental development of IEEE 1394 tree identify protocol', *Formal Aspects of Computing*, Vol. 14, No. 3, pp.215–227.

Abrial, J-R. (1996) *The B-book: Assigning Programs to Meanings*, Cambridge University Press, New York, NY, USA.

Abrial, J-R. (2001) 'Event driven distributed program construction', MATISSE project.

Abrial, J-R. (2007) 'A system development process with Event-B and the Rodin platform', in Butler, Michael, Hinchey, Michael, G. and Larrondo-Petrie, María, M. (Eds): *Formal Methods and Software Engineering*, Vol. 4789, pp.1–3, Springer Berlin Heidelberg.

Abrial, J-R. (2010) *Modeling in Event-B – System and Software Engineering*, Cambridge University Press, New York, NY.

Akyildiz, I. and Kasimoglu, I. (2004) 'Wireless sensor and actor networks: research challenges', *Ad Hoc Networks*, Vol. 2, No. 4, pp.351–367.

Back, R. and Kurki-Suonio, R. (1983) 'Decentralization of process nets with centralized control', in *Proceedings of the 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pp.131–142.

Back, R-J. and Sere, K. (1989) 'Stepwise refinement of action systems', in J.L.A. van de Snepscheut (Ed.): *Mathematics of Program Construction, 375th Anniversary of the Groningen University, International Conference*, Groningen, The Netherlands, June 26–30, Vol. 375 of *Lecture Notes in Computer Science*, Springer, pp.115–138.

Back, R.J.R. and Sere, K. (1996) 'Superposition refinement of reactive systems', *Formal Aspects of Computing*, Vol. 8, No. 3, pp.324–346.

Ball, E. and Butler, M. (2006) 'Using decomposition to model multi-agent interaction protocols in Event-B', in *FM '06 Doctoral Symposium*, Springer.

Booch, G. (2007) 'Speaking truth to power', *IEEE Software*, Vol. 24, No. 2, pp.12–13.

Cansell, D. and Méry, D. (2006) 'Formal and incremental construction of distributed algorithms: on the distributed reference counting algorithm', *Theoretical Computer Science*, Vol. 364, No. 3, pp.318–337.

Dan, L. and Danning, L. (2010) 'Towards a formal behavioral semantics for UML interactions', in *Information Science and Engineering (ISISE), 2010 International Symposium on*, pp.213–218.

Fathabadi, M., Salehi, A., Rezazadeh, A. and Butler, M. (2011) 'Applying atomicity and model decomposition to a space craft system in Event-B', in M.G. Bobaru, K. Havelund, G.J. Holzmann and R. Joshi (Eds.): *NASA Formal Methods*, Vol. 6617 of *Lecture Notes in Computer Science*, pp.328–342, Springer.

Gerhart, S., Craigen, D. and Ralston, T. (1994) 'Case study: Paris Metro signaling system', *Software, IEEE*, Vol. 11, No. 1, pp.32–35.

Gungor, V.C., Akan, O.B. and Akyildiz, I.F. (2008) 'A real-time and reliable transport (rt) 2 protocol for wireless sensor and actor networks', *IEEE/ACM Trans. Netw.*, Vol. 16, No. 2, pp.359–370 [online] http://dx.doi.org/10.1109/TNET.2007.900413.

Hoang, T., Kuruma, H., Basin, D.A. and Abrial, J-R. (2009) 'Developing topology discovery in Event-B', in *IFM*, pp.1–19.

Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A., Varpaaniemi, K., Ilic, D. and Latvala, T. (2010) ' Supporting reuse in event-b development: modularisation approach', in *ASM '10*, pp.174–188.

Iliasov, A., Troubitsyna, E., Laibinis, L. and Romanovsky, A. (2011) 'Formal derivation of a distributed program in Event-B', in *ICFEM '11*, LNCS, pp.420–436.

Imran, M., Younis, M., Said, A.M. and Hasbullah, H. (2012) 'Localized motion-based connectivity restoration algorithms for wireless sensor and actor networks', *Journal of Network and Computer Applications*, Vol. 35, No. 2, pp.844–856 [online] http://www.sciencedirect.com/science/article/pii/S1084804511002347 (accessed 22 January 2013).

Jeremy, B. and Wei, W. (2010) 'Formal analysis of BPMN models using Event-B', in S. Kowalewski and M. Roveri (Eds.): *Formal Methods for Industrial Critical Systems – 15th International Workshop, FMICS 2010*, Vol. 6371 of *Lecture Notes in Computer Science*, pp.33–49, Antwerp, Belgium, September 20–21, Springer.

Kamali, M., Sedighian, S. and Sharifi, M. (2008) 'A distributed recovery mechanism for actor-actor connectivity in wireless sensor actor networks', in *Intelligent Sensors, Sensor Networks and Information Processing, ISSNIP 2008, International Conference on*, pp.183–188.

Kamali, M., Laibinis, L., Petre, L. and Sere, K. (2010) 'Self-recovering sensor-actor networks', in *FOCLASA*, pp.47–61.

Kamali, M., Petre, L., Sere, K. and Daneshtalab, M. (2011a) 'Formal modeling of multicast communication in 3D NoCs', in *Digital System Design (DSD), 2011 14th Euromicro Conference on*, pp.634 –642.

Kamali, M., Petre, L., Sere, K. and Daneshtalab, M. (2011b) 'Refinement-based modeling of 3D NoCs', in *Fundamental of Software Engineering (FSEN), 2011 4th IPM International Conference on*.

Kamali, M., Laibinis, L., Petre, L. and Sere, K. (2012) 'Formal development of wireless sensor-actor networks', *Science of Computer Programming* [online] http://www.sciencedirect.com/science/article/pii/S0167642312000470?v=s5 (accessed 22 January 2013).

Katz, S. (1993) 'A superimposition control construct for distributed systems', *ACM Trans. Program. Lang. Syst.*, Vol. 15, No. 2, pp.337–356.

Lecomte, T. (2009) 'Applying a formal method in industry: a 15-year trajectory', in *Proceedings of the 14th International Workshop on Formal Methods for Industrial Critical Systems, FMICS '09*, Springer-Verlag, Berlin, Heidelberg, pp.26–34.

Ngai, E.C.H., Lyu, M.R. and Liu, J. (2006) 'A real-time communication framework for wireless sensor-actuator networks', in *Proc. of the IEEE Aerospace Conference, Big Sky*.

Ray, A. and Cleaveland, R. (2003) 'Architectural interaction diagrams: AIDs for system modeling', in *Software Engineering, Proceedings. 25th International Conference on*, pp.396–406.

Ray, A. and Cleaveland, R. (2004) 'Formal modeling of middleware-based distributed systems', *Electron. Notes Theor. Comput. Sci.*, Vol. 108, No. 1, pp.21–37 [online] http://dx.doi.org/10.1016/j.entcs.2004.01.010 (accessed 22 January 2013).

Rehm, J. (2010) 'Proved development of the real-time properties of the IEEE 1394 root contention protocol with the Event-B method', *International Journal on Software Tools for Technology Transfer*, Vol. 12, No. 1, pp.39–51.

Rodin Modularisation Plug-in (2011) [online] http://wiki.event-b.org/index.php/Modularisation_Plug-in/ (accessed 26 October 2011).

Rodin Tool Platform (2006) [online] http://www.event-b.org/platform.html/ (accessed 26 October 2011).

Woodcock, J., Larsen, P., Bicarregui, J. and Fitzgerald, J. (2009) 'Formal methods: practice and experience', *ACM Comput. Surv.*, Vol. 41, No. 4, pp.1–36.

# Paper III

## Topology-based Mobility Model for Wireless Networks

**Ansgar Fehnker, Peter Höfner, Maryam Kamali and Vinay Mehta**

Originally published in: K. Joshi et al. (Eds.), *Proceedings of the 10th International Conference on Quantitative Evaluation of Systems conference - QEST 13*, Lecture Notes in Computer Science Vol. 8054, pp. 389-404, Springer-Verlag, 2013.

# Topology-Based Mobility Models for Wireless Networks

Ansgar Fehnker[1], Peter Höfner[2,3], Maryam Kamali[4,5], and Vinay Mehta[1]

[1] University of the South Pacific, Fiji
[2] NICTA*, Australia
[3] University of New South Wales, Australia
[4] Turku Centre for Computer Science (TUCS), Finland
[5] Åbo Akademi University, Finland

**Abstract.** The performance and reliability of wireless network protocols heavily depend on the network and its environment. In wireless networks node mobility can affect the overall performance up to a point where, e.g. route discovery and route establishment fail. As a consequence any formal technique for performance analysis of wireless network protocols should take node mobility into account. In this paper we propose a topology-based mobility model, that abstracts from physical behaviour, and models mobility as probabilistic changes in the topology. We demonstrate how this model can be instantiated to cover the main aspects of the random walk and the random waypoint mobility model. The model is not a stand-alone model, but intended to be used in combination with protocol models. We illustrate this by two application examples: first we show a brief analysis of the Ad-hoc On demand Distance Vector (AODV) routing protocol, and second we combine the mobility model with the Lightweight Medium Access Control (LMAC).

## 1 Introduction

The performance and reliability of network protocols heavily depend on the network and its environment. In wireless networks node mobility can affect the overall performance up to a point where e.g. route discovery and route establishment fail. As a consequence any formal technique for analysis of wireless network protocols should take node mobility into account.

Traditional network simulators and test-bed approaches usually use a detailed description of the physical behaviour of a node: models include e.g. the location, the velocity and the direction of the mobile nodes. In particular changes in one of these variables are mimicked by the mobility model. It is common for network simulators to use *synthetic models* for protocol analysis [15]. In this class of models, a mobile node randomly chooses a direction and speed to travel from its current location to a new location. As soon as the node reaches the

* NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

K. Joshi et al. (Eds.): QEST 2013, LNCS 8054, pp. 389–404, 2013.
© Springer-Verlag Berlin Heidelberg 2013

new location, it randomly chooses the next direction. Although these models abstract from certain characteristics such as acceleration, they still cover most of the physical attributes of the mobile node. Two well-known synthetic mobility models are the *random walk* (e.g. [1]) and the *random waypoint model* (e.g. [2]).

However, a physical mobility model is often incompatible with models of protocols, in particular protocols in the data link and network layers, due to limitations of the used modeling language and analysis tools. Even if it could be included, it would add a high complexity and make automatic analysis infeasible. From the point of view of the protocol it is often sufficient to model changes on the topology (connectivity matrix) rather than all physical behaviour.

In this paper we propose a topology-based mobility model that abstracts from physical behaviour, and models mobility as probabilistic changes in the topology. The main idea is to identify the position of a node with its current set of neighbours and determine changes in the connectivity matrix by adding or deleting nodes probabilistically to this set. The probabilities are distilled from the random walk or the random waypoint model. The resulting model is not meant to be a stand-alone model, but to be used in combination with protocol models. For this, we provide an Uppaal template for our model, which can easily be added to existing protocol models. The paper illustrates the flexibility of our model by two application examples: the first analyses quantitative aspects of the Ad hoc On-Demand Distance Vector (AODV) protocol [14], a widely used routing protocol, particularly tailored for wireless networks; the second example presents an analysis of the Lightweight Media Access Control (LMAC) [12], a protocol designed for sensor networks to schedule communication, and targeted for distributed self-configuration, collision avoidance and energy efficiency.

The rest of the paper is organised as follows: after a short overview of related work (Sect. 2), we develop the topology-based mobility model in Sect. 3. In Sect. 4 we present a simulator that is used to compute the transition probabilities for two common mobility models. In Sect. 5, we combine the distilled probabilities with our topology-based model to create an Uppaal model. Before concluding in Sect. 7, we illustrate how the model can be used in conjunction with protocol models. More precisely we present a short analysis of AODV and LMAC.

## 2 Related Work

Mobility models are part of most network simulators such as ns-2. In contrast to this, formal models used for verification or performance analysis usually assume a static topology, or consider a few scenarios with changing topology only. For the purpose of this section, we distinguish two research areas: mobility models for network simulators and models for formal verification methods.

Mobility models for network simulators either replay traces obtained from real world, or they use synthetic models, which abstract from some details and generate mobility scenarios. There are roughly two dozen different synthetic models (see [15,4] for an overview), starting from well-known models such as the *random*

*walk model* (e.g. [1]) and the *random way point model* (e.g. [2]), via *(partially) deterministic models* and *Manhattan models* to *Gauss-Markov* and *gravity mobility models*. All these models are based on the physical behaviour of mobile nodes, i.e. each node has a physical location (in 2D or 3D[1]), a current speed and a direction it is heading to. As these models cover most of the physical behaviour, they are most often very complex (e.g. [13,10]) and include for example mathematics for Brownian motion. Due to this complexity these models cannot be incorporated directly into formal models for model-checking. This paper describes how two of these models, the random waypoint, and the random walk model, can be used to distill transition probabilities for a mobility model, which can easily be combined with formal protocol models.

Including mobility into a model for formal verification is not as common as it is for network simulators. If they are included, then typically in the protocol specification and therefore can rarely be reused for the analysis of different protocols. Moreover, formal verification often abstracts entirely from the underlying mobility model and allows arbitrary topology changes [9,5,8]. Other approaches allow only random, but very limited changes in the topology, often in the form of a scenario that involves deletion or creation of links [6,18,17]. Song and Godskesen propose in [16] a framework for modelling mobility; it models connectivity by distributions and propose a probabilistic mobility function to model mobility, without any specifics. This paper takes a similar approach, but adjacency matrices to model connectivity, and works out and analyses the transition probabilities obtained for two mobility models.

Our contribution is the following: we take the idea that the position of a mobile node can be characterised by a set of neighbours, which determines the topology, and we then define mobility as transitions between these sets. We then analyse the geometry of mobile nodes in a grid and determine which parameters actually influence the transition probabilities. In fact we found that some parameters, such as the step size of the random walk model have no influence on the transition probabilities. Based on this observation we build a topology-based mobility model which can easily be combined with protocol models.

## 3 Topology-Based Mobility Model

Our model takes up the position of the protocol: for a protocol it only matters whether data packets can be sent to a node, i.e. whether the node is within transmission range. The speed, the direction and other physical attributes are unimportant and irrelevant for the protocol. Hence the topology-based mobility model we introduce abstracts from all physical description of a node, and also largely abstracts from time. It models the node as a set of one-hop neighbours, i.e. nodes that are within transmission range of the node. Movement is modelled as a transition from one set of neighbours to another.

We assume that the node to be modelled moves within a quadratic $N \times N$-grid of stationary nodes. For simplicity we assume that nodes in the grid have a

[1] 3D is required when nodes model aerospace vehicles, such as UAVs.

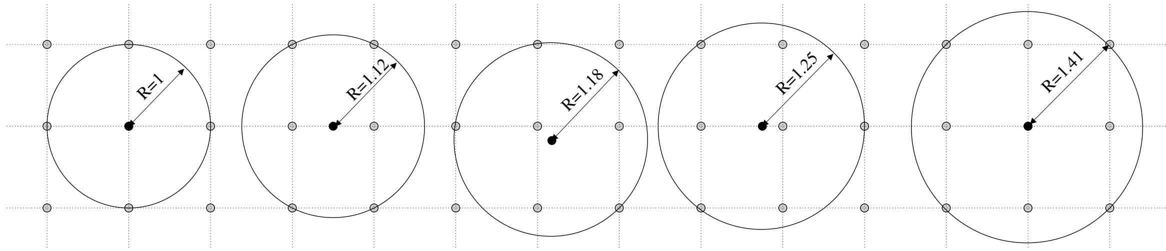



**Fig. 1.** Transmission ranges 1, $\frac{\sqrt{5}}{2} \approx 1.12$, $\frac{5}{6}\sqrt{2} \approx 1.18$, 1.25 and $\sqrt{2} \approx 1.41$

distance of 1, and that both the stationary and the mobile node have the same transmission range $R$. Obviously, the model depends on the grid size and the transmission range. We further assume that the transition range $R$ is larger than 1 and strictly smaller than $\sqrt{2}$. If it were smaller than 1 nodes in the grid would be outside of the range of all neighbours, if it were larger than $\sqrt{2}$ nodes could communicate diagonally in the grid.

The network topology of all nodes, including the mobile node, can be represented by an adjacency or *connectivity matrix* $A$ with

$$A_{i,j} = \begin{cases} 1 & \text{if } D(i,j) \leq R \\ 0 & \text{otherwise ,} \end{cases}$$

where $D(i,j)$ is the distance between the nodes $i$ and $j$ using some kind of metric, such as the Euclidean distance. While the connectivity matrix has theoretically $2^{N^2}$ possible configurations, with $N$ the number of nodes, a network with one mobile node will only reach a small fraction of those. First, the matrix is symmetric. Second, all nodes, except for one, are assumed static, and the connectivity $A_{i,j}$ between two static nodes $i$ and $j$ will be constant. Third, due to the geometry of the plane, even the mobile node can only have a limited number of configurations. For example, neither a completely connected node, nor a completely disconnected node is possible given the transmission range.

The possible topologies depend on the transmission range: the larger the range the larger the number of possible nodes that can be connected to the mobile node. Within the right-open interval $[1, \sqrt{2})$, the set of possible topologies changes at values $\frac{\sqrt{5}}{2}$, $\frac{5}{6}\sqrt{2}$ and 1.25. These values can be computed with basic trigonometry. Fig. 1 illustrates which topologies become possible at those transmission ranges.

By considering the transmission range of the stationary nodes, one can partition the plane into regions in which mobile nodes will have the same set of neighbours. The boundaries of these regions are defined by circles with radius $R$ around the stationary nodes. Fig. 2 depicts three possible regions and a transmission range $R = 1.25$; stationary nodes that are connected to the mobile node (located somewhere in the coloured area) are highlighted. As convention we will number nodes from the top left corner, starting with node 0. This partitioning abstracts from the exact location of the mobile node. Mobility can now be expressed as a change from one region to the next. The topology-based model will capture the changing topology as a Markovian transition function, that assigns to a pair of topologies a transition probability.

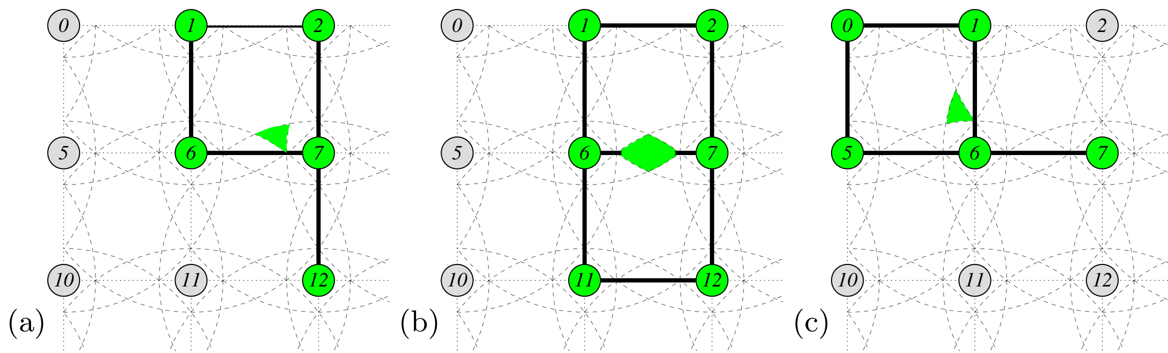



**Fig. 2.** Three regions and the corresponding set of neighbours for range $R = 1.25$

The number of possible transitions is also limited by the partition, as every region is bounded by a small number of arcs. If a mobile node transits an arc, a static node has to be added to or deleted from its set of neighbours. Consider, for example, the region that corresponds to set $\{1, 2, 6, 7, 12\}$ in Fig. 2(a). If the mobile node crosses the arc to the bottom left, node 11 will be added (Fig. 2(b)). The other two arcs of $\{1, 2, 6, 7, 12\}$ define the only two other transitions that are possible from this set.

We call a mobility model *locally defined* if congruent regions yield the same transition probabilities. Regions are congruent if they can be transformed into each other by rotation, reflection and translation. By extension we call transitions that correspond to congruent arcs in such regions also congruent. The movement of a node in a locally defined mobility model is independent from its exact position in the grid. The changes that can occur depend only on the topology of the current neighbours. For example, the congruent sets $\{1, 2, 6, 7, 12\}$ and $\{0, 1, 5, 6, 7\}$ in Fig. 2(a) and (c), would have the same transition probabilities.

In some cases this principle will uniquely determine the transition probability: the set $\{1, 2, 6, 7, 11, 12\}$ in Fig. 2(b) is bounded by 4 identical arcs. This means that all of them should correspond to a probability of $\frac{1}{4}$. For other regions the partition implies a relation/equation between some probabilities, but does not determine them completely. Considering only transitions in a single cell of the grid yields just a few and very symmetric transitions between possible topologies. Fig. 3 depicts the transitions as transitions between topologies.

One way to assign probabilities is to require that they are proportional to the length of the arc. Alternatively, probabilities may be estimated by simulations of a moving node in the plane. Note, that the resulting probabilistic transition system will be memoryless, i.e. the probability of the next transition depends only on the current region (set of neighbours). In the next section, we will see that the common random waypoint model is not locally defined, i.e. the local topology is not sufficient to determine the transition probabilities.

## 4 Simulations of Two Mobility Models

In the previous section we proposed a topology-based mobility model, based on transition probabilities; the exact values for the probabilities, however, were

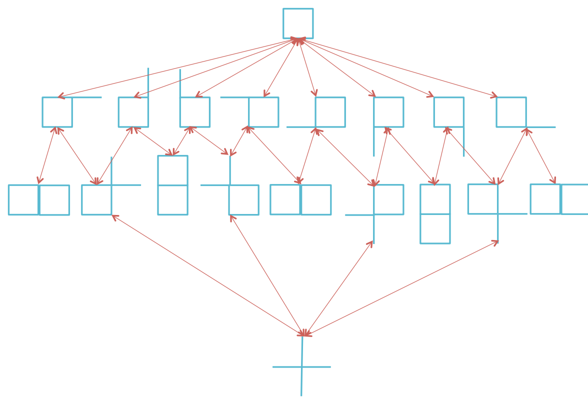


**Fig. 3.** Possible transitions within a single grid cell for $R = 1.25$

not specified. In this section we use a simulator to compute it for two common mobility models, a random walk model, and a random waypoint model.

### 4.1 Simulator

The simulator considers a single mobile node in an $N \times N$ grid of stationary nodes. As before, we assume a distance of 1 between the nodes on the grid. The initial position $(x_0, y_0)$ of the mobile node is determined by a uniform distribution over $[0, N-1] \times [0, N-1]$, i.e. $x_0 \sim \mathcal{U}([0, N-1])$ and $y_0 \sim \mathcal{U}([0, N-1])$. Depending on the mobility model chosen, the simulator then selects a finite number of waypoints $(x_1, y_1), \ldots, (x_n, y_n)$, and moves along a straight line from waypoint $(x_i, y_i)$ to the next $(x_{i+1}, y_{i+1})$.

The *random waypoint* model uses a uniform distribution over the grid to select the next way point, i.e. for all $x_i$, we have $y_i$, $x_i \sim \mathcal{U}([0, N-1])$ and $y_i \sim \mathcal{U}([0, N-1])$. The choice of the next waypoint is independent of the previous waypoint. This model is the most common model of mobility for network simulators, even if its merits have been debated [19]. A consequence of the waypoint selection is that the direction of movement is not uniformly distributed; nodes tend to move more towards the centre of the square interval.

As an alternative we are using a simple *random walk* model. Given way point $(x_i, y_i)$ the next way point is computed by $(x_i, y_i) + (x_\Delta, y_\Delta)$ where both $x_\Delta$ and $x_\Delta$ are drawn from a normal distribution $\mathcal{N}(0, \sigma)$. This also means that the Euclidean distance between waypoints $||(x_\Delta, y_\Delta)||$ has an expected value of $\sigma$, which defines the average step size in the random walk model. By this definition, the model is unbounded, i.e. the next waypoint may lie outside the grid. If this happens the simulator computes the intersection of the line segment with the grid's boundary and reflects the waypoint at that boundary. In this model the mobile node moves from the first waypoint to the boundary, and from there to the reflected waypoint. For the purposes of this paper the intersections with the boundary do not count as waypoints.

Since the topology-based mobility model introduced in Sect. 3 abstracts from acceleration and speed, these aspects are not included in the simulation either. The simulator checks algebraically for every line segment from $(x_i, y_i)$ to

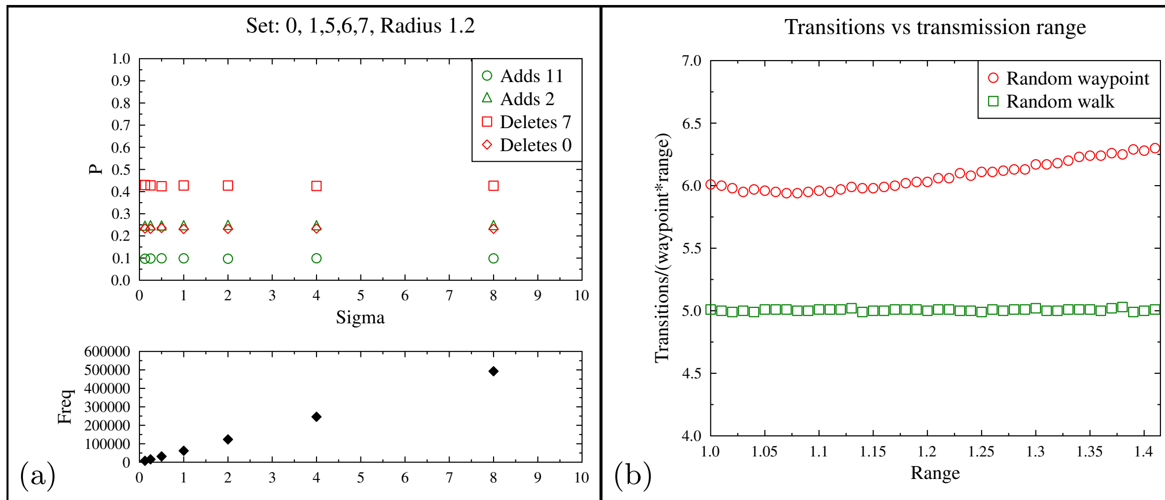



**Fig. 4.** (a) Transition probabilities and occurrences of set $\{0, 1, 5, 6, 7\}$. (b) The relation between number of transitions, the number of waypoints, and the transmission range.

$(x_{i+1}, y_{i+1})$ if it intersects with a node's transmission range $R$ (given by a circle with radius $R$ and the node in its centre). The simulator sorts all the events of nodes entering and leaving the transmission range and computes a sequence of sets of neighbours. This sequence is then used to count occurrences of transitions between these sets that are used to compute relative transition probabilities.

### 4.2 Simulation Results

The simulator is implemented in C++, and used to generate transition probabilities for the topology-based mobility model of Section 3. The simulator allows also a more detailed analysis of these two mobility models, in particular how the choice of parameters (grid size, transmission range, and standard deviation of the normal distribution $\sigma$) affects the transition probabilities. In this section we discuss some results for scenarios with a single mobile node on a $5 \times 5$ grid.

The simulation of the random walk model demonstrates a few important invariants. One observation is that the transition probabilities do not depend on the size of $\sigma$. This fact is illustrated by Fig. 4(a). The top part of this figure shows the probabilities that certain nodes are added or deleted from the set $\{0, 1, 5, 6, 7\}$. While $\sigma$ ranges from $\frac{1}{8}$ to 8 the probabilities remain constant. The bottom part of the figure depicts the frequency with which the set occurs. Here there is a linear relation between $\sigma$ and the total number of times that the set is visited. This is explained by the fact that $\sigma$ is also the average step size, and doubling it means that twice as many transitions should be taken along the path.

Another linear relation exists between the total number of transitions along a path and the transmission range (cf. Fig. 4(b)). This relation is explained by the fact that the length of the boundary of each transmission area is linear to the range. For $\sigma = 1$, and $R = 1$, approximately 5 transitions will occur between any two waypoints. The ratio transition/range is constant for an increasing range. Note, that this number is independent of the grid size, and grows linearly with $\sigma$.

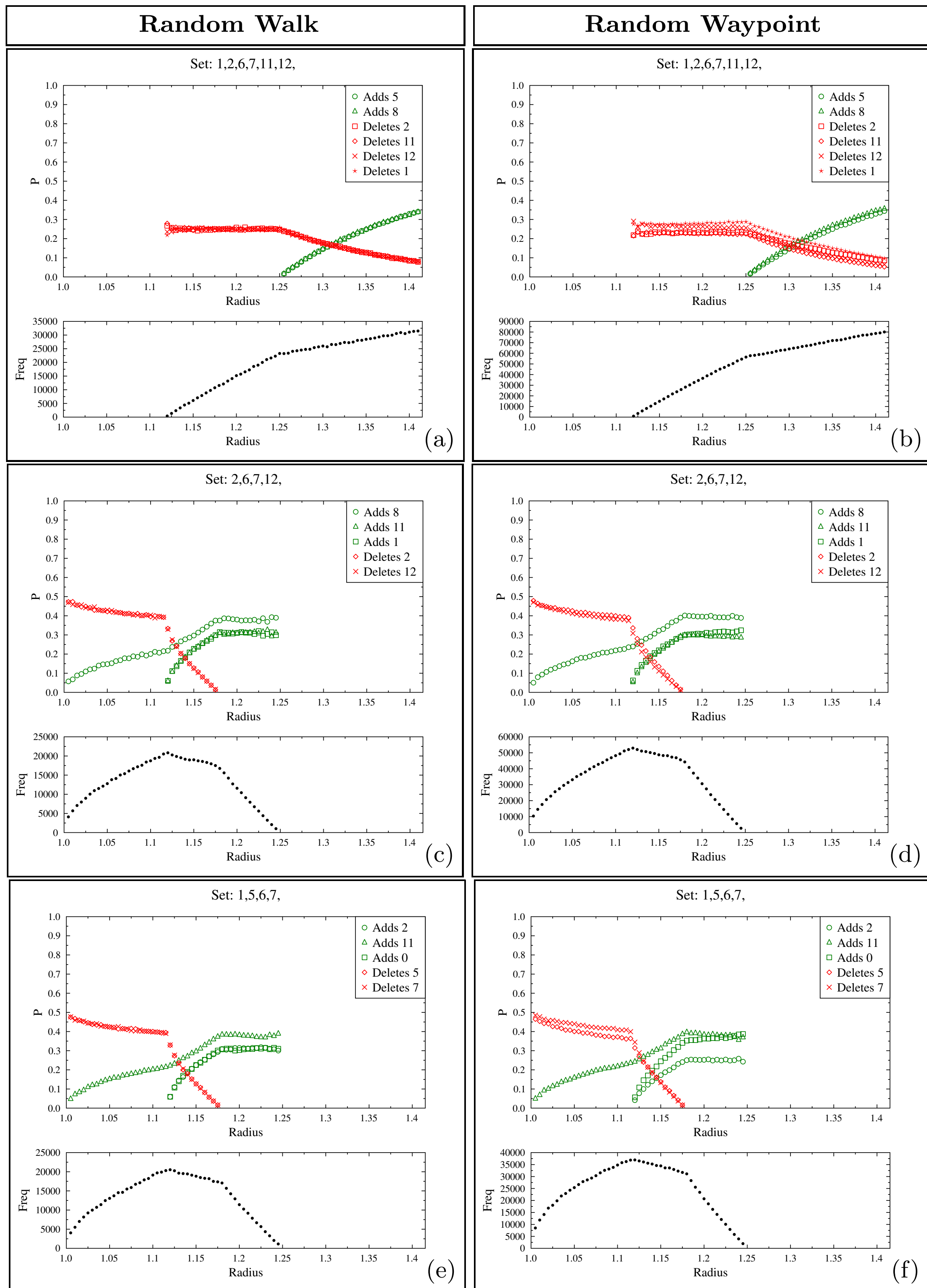



**Fig. 5.** Selected simulation results of the random walk and the random waypoint model

These invariants do not hold for the random waypoint model. The ratio of transitions to range is not constant, as illustrated in Fig. 4(b). This is because transitions are not evenly distributed but cluster towards the center of the grid. The ratio is also dependent on the size of the grid. In a larger grid the distance between waypoints will be larger, and more transitions occur per waypoint.

For the random walk model we found that the step size $\sigma$ has no effect on the actual transition probabilities. The effect of the transmission range on the transition probabilities is less trivial. Fig. 5 shows a few illustrative examples. Similar result were obtained for all possible sets of neighbours.

Fig. 5(a) depicts the results for $\{1, 2, 6, 7, 11, 12\}$, a set of six nodes that form a rectangle. This set cannot occur if the transmission ranges are smaller than $\frac{\sqrt{5}}{2}$ (cf. Sect. 3). For transmission ranges $R \in [\frac{\sqrt{5}}{2}, 1.25]$ the only possible transitions are to delete one of the four vertices located at the corners of the rectangle. In the random walk model the probability for these four transitions is $\frac{1}{4}$. Fig. 5(a) also illustrates that for transmission ranges $R \geq 1.25$, it is possible to add one additional node (either 5 or 8), reaching a set with 7 one-hop neighbours. As the range increases, the probability of this happening increases. At the same time the probability of deleting a vertex decreases.

Fig. 5(b) consider the same set of neighbours as Fig. 5(a), but under the random waypoint model. It demonstrates that this model is not locally defined, as congruent transitions, e.g. deleting vertices, do not have the same probability. The probability also depends of the distance of a node to the centre of the grid.

Fig. 5(c–f) show the transition probabilities for sets of neighbours that occur only if $R \in [1, 1.25]$: if $R < 1$, the transmission range is too small to cover the sets $\{2, 6, 7, 12\}$ and $\{1, 5, 6, 7\}$, resp.; if $R > 1.25$ the transmission range of the mobile will always contain more than four nodes. The observation is that as the transmission range increases, the probability of deleting a node decreases, while the probability of adding nodes increases. The sets $\{2, 6, 7, 12\}$ and $\{1, 5, 6, 7\}$ have the same basic "⊤" shape; one is congruent to the other. Hence, for the random walk model both sets have essentially the same transition probability; but also the frequency with which the sets occur is the same. This confirms that the position or orientation in the grid does not matter.

For the random waypoint model this no longer holds. The transition probabilities of similarly shaped neighbourhoods are not similar, but also determined by the position relative to the centre: the closer the set is to the centre the often it occurs in paths. Note, Fig. 1(d) and (f) use different scales for the frequency.

To conclude this section, we summarise our findings:

**Random walk model:**

- The transition probabilities are independent of $\sigma$ and the grid size;
- The number of transitions per waypoint path grows linear with the range;
- The transition probabilities of congruent transitions are the same;
- The probabilities depend only locally on the set of nodes within range.

**Random waypoint model:** None of the above observations hold.

**Table 1.** Number of possible topologies, in relation to the range and the grid size[2]

| Grid size | Transmission range | | | | | |
|---|---|---|---|---|---|---|
| | $[1,1]$ | $(1,1.12)$ | $(1.12,1.18)$ | $(1.18,1.25)$ | $[1.25,1.25]$ | $(1.25,1.41)$ |
| $2\times 2$ | 9 | 9 | 5 | 5 | 5 | 5 |
| $3\times 3$ | 32 | 41 | 49 | 49 | 37 | 41 |
| $4\times 4$ | 69 | 97 | 133 | 133 | 101 | 117 |
| $5\times 5$ | 120 | 177 | 257 | 257 | 197 | 233 |
| $6\times 6$ | 185 | 281 | 421 | 421 | 325 | 389 |
| $7\times 7$ | 264 | 409 | 625 | 625 | 485 | 585 |
| $8\times 8$ | 357 | 561 | 869 | 869 | 677 | 821 |
| $9\times 9$ | 464 | 737 | 1153 | 1153 | 901 | 1097 |
| $10\times 10$ | 585 | 937 | 1477 | 1477 | 1157 | 1413 |

## 5 Uppaal Model

This section describes an Uppaal model that implements the topology-based mobility model described in Sect. 3, and uses the transition probabilities obtained in Sect. 4. The model is not meant to be stand-alone, but meant to be used within other protocol models. It assumes that an adjacency matrix `bool topoloy[N][N]` is used. The constant `N` is the size of the grid plus the mobile node. Depending on whether the random walk or random waypoint model is used, the model includes parameters for grid size and transmission range.

The template provides a list of all possible sets of neighbours. Table 1 shows the numbers of possible sets depending on the size of the grid and the transmission range. The results show that even for relatively large grids the number of possible sets of neighbors of the mobile node is limited. They will increase the potential state space only by three order of magnitude. The reachable space may increase by more when a template for mobility is added, because the protocol might reach more states than it did for static topologies.

The Uppaal template of Fig. 6 implements a lookup table of transition probabilities. After initialisation the template loops through a transition that changes the topology probabilistically. It contains a clock `t`, a guard `t>=minframe` and an invariant `t<=maxframe` to ensure that the change happens once in the interval `[minframe,maxframe]`. The values of `minframe` and `maxframe` determine the frequency of topology changes, and hence simulate the speed of a node.

The lookup is implemented by functions `updatemapindex`, `changeprob` and `changenode`. After every topology change, the function `updatemapindex` maintains the index (`mapindex`); this index into the list of possible sets is used to look up transition probabilities for a smaller set of representative sets of neighbours. Every set of neighbours is congruent to one of these representative sets. This information is used by `changeprob` to look up for a given node $i$ the probability that it will be added or deleted from the current set of neighbours. Function `changenode` implements that change.

[2] Results for the point intervals containing $\frac{\sqrt{5}}{2}$ and $\frac{5}{6}\sqrt{2}$ are omitted.

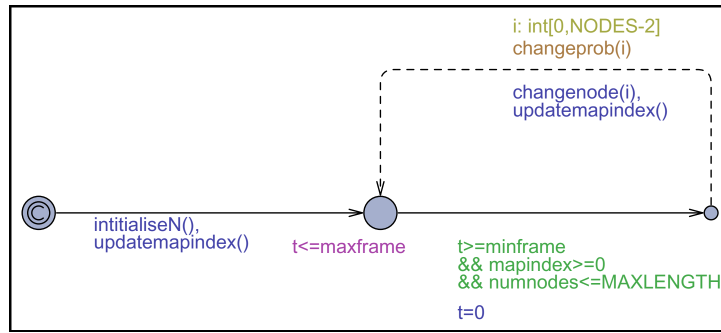



**Fig. 6.** Uppaal template for the mobility model

# 6 Application Examples

In this section we illustrate how the topology-based model can be used in combination with protocol models: first we briefly present an analysis of the Ad-hoc On demand Distance Vector (AODV) routing protocol, and second we combine the mobility model with the Lightweight Medium Control (LMAC). A detailed study of these protocols is out of the scope of the paper; we only show the applicability and power of the introduced mobility model.

Since we are interested in quantitative properties of the protocols, we are not using "classical" Uppaal, but SMC-Uppaal, the statistical extension of Uppaal [3]. *Statistical Model Checking (SMC)* [20] combines ideas of model checking and simulation with the aim of supporting quantitative analysis as well as addressing the size barrier that currently prevents useful analysis of large models. SMC trades certainty for approximation, using Monte Carlo style sampling, and hypothesis testing to interpret the results. Parameters setting thresholds on the probability of false negatives and on probabilistic uncertainty can be used to specify the statistical confidence on the result. For this paper, we choose a confidence level of 95%.

## 6.1 The Ad-Hoc on Demand Distance Vector (AODV) Protocol

AODV is a reactive routing protocol, which means that routes are only established on demand. If a node $S$ needs to send a data packet to node $D$, but currently does not know a route, it buffers the packet and initiates a route discovery process by broadcasting a route request message in the network. An intermediate node $A$ that receives this message stores a route to $S$, and re-broadcasts the request. This is repeated until the message reaches $D$, or alternatively a node with a route to $D$. In both cases, the node replies to the route request by unicasting a route reply back to the source $S$, via the previously established route.

An Uppaal model of AODV is proposed in [6]. The analysis performed on this model was done for static topologies and for topologies with very few changes. This limits the scope of the performance analysis. Here, the mobility automaton is added to the model of AODV. Since the mobility automaton is an almost independent component, it can be easily integrated into any Uppaal model that model topologies by adjacency matrices.

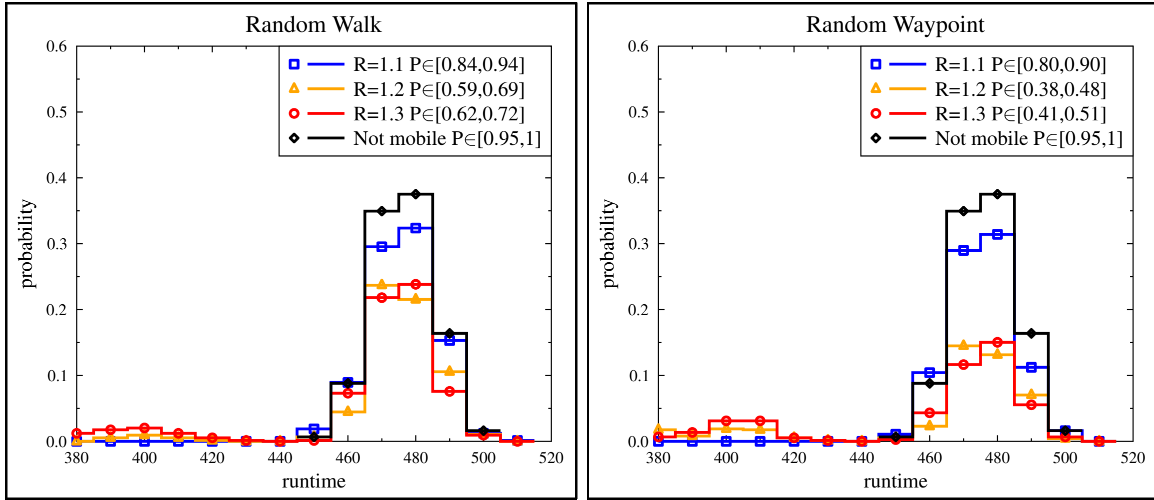



**Fig. 7.** AODV: probability of packet delivery within a certain time

Our experiments consider scenarios with a single mobile node moving within a $4 \times 4$ grid. A data packet destined for a randomly chosen stationary node is injected at a different stationary node. During route discovery the mobile node will receive and forward route requests and replies, as any other node will do.

The experiment determines the probability that the originator of the route request learns a route to the destination within 2000 time units. This time bound is chosen as a conservative upper bound to ensure that the analyser explores paths to a depth where the protocol is guaranteed to have terminated. In (SMC-) Uppaal syntax this property can be expressed as

$$\texttt{Pr[<=2000](<> node(OIP).rt[DIP].nhop!=0)} \,. \tag{1}$$

The variable `node(OIP).rt` denotes the routing table of the originator `OIP`, and the field `node(OIP).rt[DIP].nhop` represents the next hop on the stored route to the destination `DIP`. In case it is not `0`, a route to `DIP` was successfully established. The property was analysed for the random walk and the random waypoint model with three different transmission ranges $R$: 1.1, 1.2, and 1.3. SMC-Uppaal returns a probability interval for the property (1), as well as a histogram of the probabilities of the runtime needed until the property is satisfied.

The results are presented in Fig. 7. The legend contains, besides the name of the model, the probability interval. For example, the random walk model with $R = 1.1$ satisfies property (1) with a probability $P \in [0.84, 0.94]$. In contrast to that the probability of route establishment in a scenario without a mobile node is $[0.95, 1]$, which indicates that the property is always satisfied. The probability intervals show that all scenarios with a mobile node have a lower probability for route discovery, some dramatically so. The random waypoint model with $R = 1.2$ has a probability interval of $[0.38, 0.48]$, which means that more than half of all route discovery processes fail. It is also notable that the random walk models have better results than the corresponding random waypoint models. Finally, the mobility models with $R = 1.1$ have a significantly higher probability to succeed than the other four models with $R = 1.2$ and $R = 1.3$.

The histograms show another interesting finding. The time it takes for a route reply to be delivered, if it is delivered, can be shorter for the models with the mobile node. Apparently, the mobile node can function as a messenger between originator and destination; not just by forwarding messages, but also by physically creating shortcuts.

### 6.2 The Lightweight Medium Access Control (LMAC) Protocol

LMAC [11] is a lightweight time division medium access protocol designed for sensor networks to schedule communication, and targeted for distributed self-configuration, collision avoidance and energy efficiency. It assumes that time is divided into frames with a fixed number of time slots. The purpose of LMAC is to assign to every node a time slot different from its one- and two-hop neighbours. If it fails to do so, collisions may occur, i.e. a node receives messages from two neighbours at the same time. However, LMAC contains a mechanism to detect collisions and report them to the nodes involved, such that they choose (probabilistically) a new time slot.

A (non-probabilistic) Uppaal model for LMAC was developed in [7], where it was also used to study static topologies. Based on this model a probabilistic model was developed [11]. This model was then used to study the performance of LMAC for heuristically generated topologies with 10 nodes [3]. The model we use for this paper differs in one aspect from [3]: it uses a smaller frame, with only six time slots, rather than 20. The purpose of LMAC is to assign time slots such that collisions are avoided or resolved, even if the number of time slots is restricted. For a $3 \times 3$ grid, it is possible to find a suitable assignment with only five time slots; six time slots should therefore be sufficient to cover a network with 10 nodes (one mobile node), although it might be challenging.

We check the following two properties:

```
Pr[<=2000](<> forall (i: int[0,9]) slot_no[i]>=0)      (2)
Pr[collisions<=2000](<> time>=2000) .                   (3)
```

The first property holds if, at some time point (before time 2000), all nodes are able to select a time slot. While this does not guarantee the absence of collisions, it does guarantee that all nodes have been able to participate in the protocol. The second property checks whether it is possible to reach 2000 time units, with less than 2000 collisions. This property is true for all runs. It is used merely to obtain a histogram of the number of collisions.

The results are illustrated in the histogram of Fig. 8: for all models with a mobile node, the property (2) is satisfied (the probability interval is $[0.95, 1]$, by a confidence level of 95%). The detailed results show that all runs reach a state in which all nodes have chosen a time slot. For the model without a mobile node, the probability interval is $[0.80, 0.90]$. This means that in at least 10% of all cases LMAC is not able to assign a time slot to all nodes; the histogram shows runs with 80–90, 160–170, and 240–250 collisions. These are runs in which one, or more nodes are engaged in a perpetual collision. Interestingly, this type of perpetual collisions do not occur in models with a mobile node. The mobile

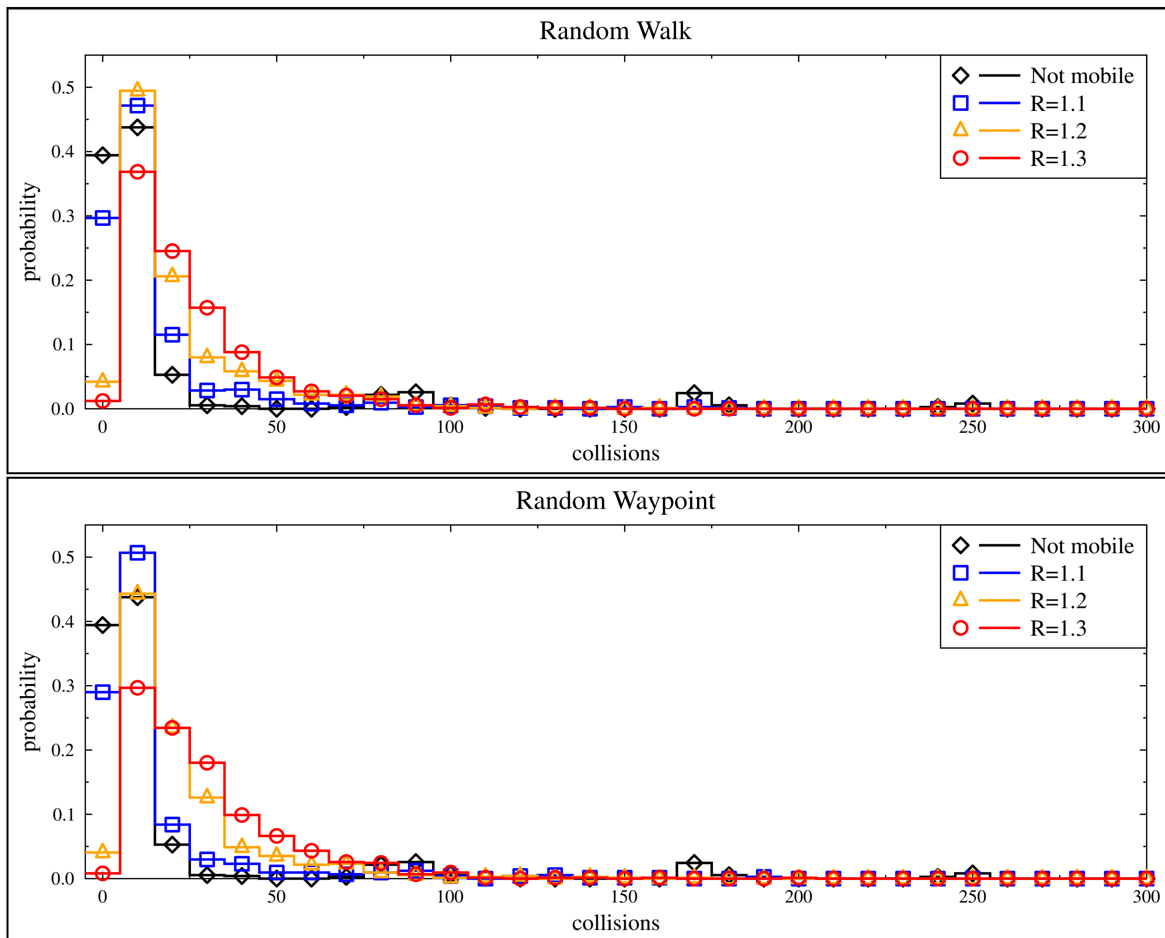



**Fig. 8.** LMAC: number of collisions within 2000 time units

node functions as an arbiter, which, as it moves around, detects and reports collisions that static nodes could not resolve.

The histograms reveal a few other interesting findings. In the model without mobility about 40% of the runs have no collisions. For both mobility models with transmission range $R = 1.1$ this drops to about 30%. For larger transmission ranges this drops even further to close to 0%, which means that almost all runs have at least some collisions. The differences between range $R = 1.1$, $R = 1.2$ and $R = 1.3$ is explained by the fact that the mobile node for $R = 1.1$ will have at most 5 neighbours, while for $R = 1.3$ it may be 7 neighbours. A larger neighbourhood makes choosing a good time slot more difficult. This is confirmed by another observation, namely that for $R = 1.1$ only a few runs have more than 20 collisions (approx. 12% of the runs, both random walk and random waypoint), while for a range of 1.2 and 1.3 it is in the range from 25% to 45%.

Both application examples show that introducing mobility can change the behaviour of network protocols significantly. As mentioned above, the purpose of these application examples was not to analyse these protocols in detail, but to show that the topology-based mobility models can be used to improve the scope of performance analyses of such protocols.

## 7 Conclusion

In this paper we have proposed an abstract, reusable, topology-based mobility model for wireless networks. The model abstracts from all physical aspects of a node as well as from time, and hence results in a simple probabilistic model. To choose a right level of abstraction, we have studied possible transitions and configurations of network topologies. To determine realistic transition probabilities regarding existing mobility models, we have performed simulation-based experiments. In particular, we have distilled probabilities for the random walk and the random waypoint model (using different transition ranges). We have then combined the topology-based model with the distilled probabilities and have created a (SMC-)Uppaal model[3]. The generated model is small and can easily be combined with other Uppaal models specifying arbitrary protocols. To illustrate this claim we have combined our model with a model of AODV and LMAC, resp. By this we were able to demonstrate that topology-based mobility models can be used to improve the scope of performance analysis of such protocols.

There are several possible directions for future work. First, we hope that our model is combined with a variety of protocols. Anybody who has some experience with the model checker Uppaal should be able to integrate our model easily. Second, we want to extend our mobility model to more than one mobile node. Having many mobile nodes will most likely increase the state space significantly, but statistical model checking should overcome this drawback. Last, but not least, we plan to use the mobility model to perform a thorough and detailed analysis of AODV and LMAC. In this paper we have only scratched the surface of the analysis; we expect to find unexpected behaviour in both protocols.

## References

1. Basu, P., Redi, J., Shurbanov, V.: Coordinated flocking of UAVs for improved connectivity of mobile ground nodes. In: MILCOM 2004, pp. 1628–1634. IEEE (2004)
2. Bettstetter, C., Hartenstein, H., Pérez-Costa, X.: Stochastic properties of the random waypoint mobility model. Wireless Networks 10(5), 555–567 (2004)
3. Bulychev, P., David, A., Larsen, K., Mikučionis, M., Poulsen, D.B., Legay, A., Wang, Z.: UPPAAL-SMC: Statistical model checking for priced timed automata. In: Wiklicky, H., Massink, M. (eds.) Quantitative Aspects of Programming Languages and Systems. EPTCS, vol. 85, pp. 1–16. Open Publishing Association (2012)
4. Camp, T., Boleng, J., Davies, V.: A survey of mobility models for ad hoc network research. In: Wireless Communications & Mobile Computing (WCMC 2002), pp. 483–502 (2002)
5. Fehnker, A., van Glabbeek, R., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: A process algebra for wireless mesh networks. In: Seidl, H. (ed.) ESOP 2012. LNCS, vol. 7211, pp. 295–315. Springer, Heidelberg (2012)
6. Fehnker, A., van Glabbeek, R., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: Automated analysis of AODV using UPPAAL. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 173–187. Springer, Heidelberg (2012)

[3] The models are available at `http://repository.usp.ac.fj/5880`

7. Fehnker, A., van Hoesel, L., Mader, A.: Modelling and verification of the LMAC protocol for wireless sensor networks. In: Davies, J., Gibbons, J. (eds.) IFM 2007. LNCS, vol. 4591, pp. 253–272. Springer, Heidelberg (2007)
8. Ghassemi, F., Ahmadi, S., Fokkink, W., Movaghar, A.: Model checking mANETs with arbitrary mobility. In: Arbab, F., Sirjani, M. (eds.) FSEN 2013. LNCS, vol. 8161. Springer, Heidelberg (2013)
9. Godskesen, J.C.: A calculus for mobile ad hoc networks. In: Murphy, A.L., Vitek, J. (eds.) COORDINATION 2007. LNCS, vol. 4467, pp. 132–150. Springer, Heidelberg (2007)
10. Groenevelt, R., Altman, E., Nain, P.: Relaying in mobile ad hoc networks: the Brownian motion mobility model. Wireless Networks 12(5), 561–571 (2006)
11. van Hoesel, L., Havinga, P.: A lightweight medium access protocol (LMAC) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. In: Networked Sensing Systems, INSS 2004, pp. 205–208. Society of Instrument and Control Engineers (SICE) (2004)
12. van Hoesel, L.: Sensors on speaking terms: schedule-based medium access control protocols for wireless sensor networks. Ph.D. thesis, University of Twente (2007)
13. McGuire, M.: Stationary distributions of random walk mobility models for wireless ad hoc networks. In: Mobile Ad Hoc Networking and Computing (MobiHoc 2005), pp. 90–98. ACM (2005)
14. Perkins, C., Royer, E.: Ad-hoc On-Demand Distance Vector Routing. In: 2nd IEEE Workshop on Mobile Computing Systems and Applications, pp. 90–100 (1999)
15. Roy, R.R.: Handbook of Mobile Ad Hoc Networks for Mobility Models. Springer (2011)
16. Song, L., Godskesen, J.C.: Probabilistic mobility models for mobile and wireless networks. In: Calude, C.S., Sassone, V. (eds.) TCS 2010. IFIP AICT, vol. 323, pp. 86–100. Springer, Heidelberg (2010)
17. Tschirner, S., Xuedong, L., Yi, W.: Model-based validation of qos properties of biomedical sensor networks. In: Embedded Software (EMSOFT 2008), pp. 69–78. ACM (2008)
18. Wibling, O., Parrow, J., Pears, A.N.: Automatized verification of ad hoc routing protocols. In: de Frutos-Escrig, D., Núñez, M. (eds.) FORTE 2004. LNCS, vol. 3235, pp. 343–358. Springer, Heidelberg (2004)
19. Yoon, J., Liu, M., Noble, B.: Random waypoint considered harmful. In: Joint Conference of the IEEE Computer and Communications (INFOCOM 2003). IEEE (2003)
20. Younes, H.: Verification and Planning for Stochastic Processes with Asynchronous Events. Ph.D. thesis, Carnegie Mellon University (2004)

# Paper IV

## Quantitative Analysis of AODV and its Variants on Dynamic Topologies using Statistical Model Checking

**Peter Höfner and Maryam Kamali**

Originally published in: V. Braberman and L. Fribourg (Eds.), *Proceedings of the 11th International Conference on Formal Modeling and Analysis of Timed Systems - FORMATS 13*, Lecture Notes in Computer Science Vol. 8053, pp. 121-136, Springer-Verlag, 2013.

# Quantitative Analysis of AODV and Its Variants on Dynamic Topologies Using Statistical Model Checking

Peter Höfner[1,4] and Maryam Kamali[2,3]

[1] NICTA, Australia
[2] Turku Centre for Computer Science (TUCS), Finland
[3] Åbo Akademi University, Finland
[4] University of New South Wales, Australia

**Abstract.** Wireless Mesh Networks (WMNs) are self-organising ad-hoc networks that support broadband communication. Due to changes in the topology, route discovery and maintenance play a crucial role in the reliability and the performance of such networks. Formal analysis of WMNs using exhaustive model checking techniques is often not feasible: network size (up to hundreds of nodes) and topology changes yield state-space explosion. Statistical Model Checking, however, can overcome this problem and allows a quantitative analysis.

In this paper we illustrate this by a careful analysis of the Ad hoc On-demand Distance Vector (AODV) protocol. We show that some optional features of AODV are not useful, and that AODV shows unexpected behaviour—yielding a high probability of route discovery failure.

## 1 Introduction

Route finding and route maintenance are critical for the performance of networks. Efficient routing algorithms become even more important when mobility of network nodes lead to highly dynamic and unpredictable environments. The Ad hoc On-Demand Distance Vector (AODV) routing protocol [15] is such an algorithm. It is widely used and particularly designed for Wireless Mesh Networks (WMNs), self-organising ad-hoc networks that support broadband communication.

Formal analysis of routing protocols is one way to systematically analyse protocols for flaws and to present counterexamples to diagnose them. It has been used in locating problems in automatic route-finding protocols, e.g. [1,4]. These analyses are performed on tiny static networks (up to 5 nodes). However, formal validation of protocols for WMNs remains a challenging task: network size (usually dozens, sometimes even hundreds of nodes) and topology changes yield an explosion in the state space, which makes exhaustive model checking (MC) techniques infeasible. Another limitation of MC is that a quantitative analysis is often not possible: finding a shortcoming in a protocol is great but does not show how often the shortcoming actually occurs.

V. Braberman and L. Fribourg (Eds.): FORMATS 2013, LNCS 8053, pp. 121–136, 2013.
© Springer-Verlag Berlin Heidelberg 2013

Statistical model checking (SMC) [19,18] is a complementary approach that can overcome these problems. It combines ideas of model checking and simulation with the aim of supporting quantitative analysis as well as addressing the size barrier. SMC trades certainty for approximation, using Monte Carlo style sampling, and hypothesis testing to interpret the results.

In this paper we demonstrate that SMC can be used for formal reasoning of routing protocols in WMNs. We perform a careful analysis of different versions of the AODV protocol. In particular, we analyse how dynamic topologies can affect the protocol behaviour. In other words, we analyse the performance of the protocol while the network topology evolves. We show that some optional features provided by AODV should be avoided since they affect the performance of the protocol. Moreover, we show that in some scenarios the behaviour of AODV is not as intended yielding a high probability of route discovery failure. When possible we suggest improvements of the protocol.

The paper is organised as follows: in Sect. 2 we give an overview of AODV, present optional features such as the resending of route requests, and sketch the encoding of AODV in SMC-Uppaal, the statistical extension of Uppaal. In Sect. 3 we describe the mobility model, which is used for our analysis of AODV. Sect. 4 discusses the experiments performed, the main contribution of this paper: (i) We show that a single mobile node can have a massive impact on the success of route discovery. Moreover we show that some options of AODV should not be used in combination, unless the protocol specification is adapted (changed). (ii) A second category of experiments reveals a surprising observation: adding "noise" (for example an additional data packet) to a network can increase the success of route discovery. (iii) The third category discusses the consequences of different speeds of mobile nodes. The paper closes with a discussion of related work in Sect. 5 and a short outlook in Sect. 6.

## 2 AODV, Its Variants and Their Uppaal Models

### 2.1 The Basic Model

The AODV routing protocol [17] is a widely used routing protocol, particularly tailored for WMNs. It is currently standardised by the IETF MANET working group and forms the basis of new WMN routing protocols, including HWMP in the upcoming IEEE 802.11s wireless mesh network standard [12].

AODV is a reactive protocol, meaning that a route discovery process is only initiated when a node $S$ in the network has to send data to a destination $D$ for which it does not have a valid entry in its own routing table. The route discovery process starts with node $S$ broadcasting a route request (RREQ) message, which is received by all nodes within $S$'s transmission range. If a node, which is different to the destination, receives a RREQ message and does not have a valid entry for the destination in its routing table, the request is forwarded by re-broadcasting the RREQ message. During this forwarding process, the intermediate node updates its routing table and adds a "reverse route" entry with destination $S$ into its routing table, indicating via which next hop the node $S$ can

be reached, and the distance in number of hops. To avoid unnecessary message sending each RREQ has a unique identifier which allows nodes to ignore RREQ messages that they have handled before.

As soon as the RREQ is received by the destination itself or by a node that knows a valid route to the destination, a route reply (RREP) is generated. In contrast to RREQ messages, a RREP message is unicast, i.e., it is only sent to a single node, not to all nodes within transmission range. The RREP message travels from its generator (either $D$ or an intermediate node knowing a route to $D$) back along the established route towards $S$, the originator of the RREQ message. All intermediate nodes on the selected route will process the RREP message and, in most cases, forward it towards $S$. However, there are scenarios where RREP message are discarded (see below). By passing a RREP message towards $S$, a node adds a "forward route" entry to its routing table.

The route discovery process is completed when the RREP reaches node $S$; an end-to-end route from $S$ to $D$ has been established, and data packets can start to flow. If any link breaks down (e.g. by a node moving out of transmission range), the node that detects the break broadcasts a route error (RERR) message.[1] All notified nodes invalidate their routing table entries that use the broken link and forward the RERR message if necessary.

Full details can be found in RFC 3561 [15], the de facto standard of AODV.

### 2.2 Variants of AODV

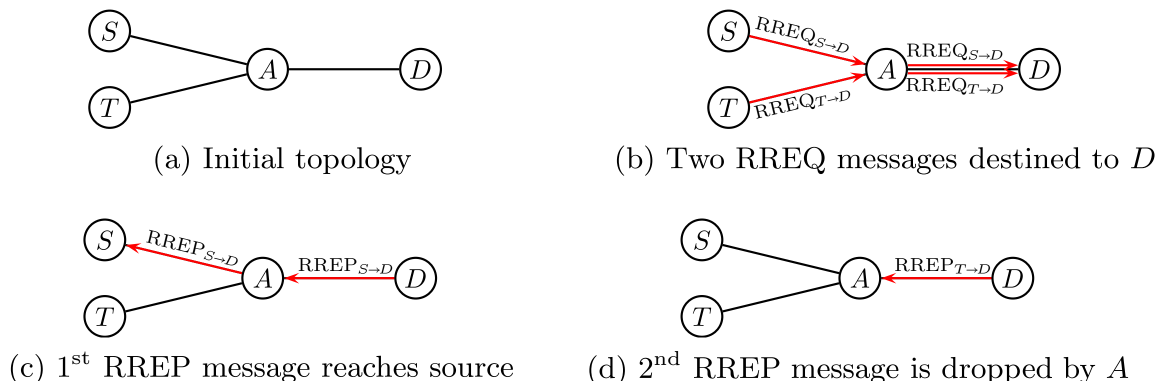
The specification of AODV [15] offers optional features, which yield different variants of the routing protocol. One aim of this paper is to compare versions of AODV with different features turned on.

**Destination Only (D) Flag.** Each RREQ message contains a field called *destination only flag*. If the value of this Boolean flag is `true`, it indicates that only the destination node is allowed to respond to this RREQ. That means that the RREQ travels through the entire network until it reaches the destination. Only then a reply is sent back. By this a *bi-directional link* between the source and the destination is (usually) established.

**Resending a Route Request.** The basic version of AODV, as presented in the previous section, suffers the problem that some routes, although they do exist, are not discovered. Reasons for route discovery failure can be message transmission failures (the receiver of a unicast message has moved out of transmission range) or the dropping of RREP messages, that should be forwarded. With respect to the latter, the problem is that a node only forwards a RREP message if it is not the originator node, *and* it has created or updated a routing table entry to the destination node described in the RREP message. [15]

---

[1] Following the RFC, a node uses precursor lists to store those nodes that are interested in some particular routes—when sending an RERR message only those neighbours are informed. However, precursor lists do not contain all neighbours that are interested in a particular route (e.g. [8]); that is why we model an improved version of AODV where RERR messages are broadcast.




(a) Initial topology

(b) Two RREQ messages destined to $D$

(c) 1st RREP message reaches source

(d) 2nd RREP message is dropped by $A$

**Fig. 1.** Route Discovery Failure

An example for route discovery failure, taken from [10], is sketched in Fig. 1.[2] On the 4-node topology depicted in Part (a) nodes $S$ and $T$, resp., initiate a route discovery process to search for a route to $D$. The messages travel through the network and reach the destination $D$ (Part (b)). We assume that $\text{RREQ}_{S \to D}$, the request stemming from $S$, reaches node $D$ first. In Part (c), $D$ handles $\text{RREQ}_{S \to D}$, creates an entry for $S$ in its routing table[3] and unicasts a RREP message to $A$. Node $A$ updates its routing table (creates an entry for $D$) and forwards the message to the source $S$. In Part (d), $D$ handles $\text{RREQ}_{T \to D}$, creates an entry for $T$ in its routing table and unicasts a RREP message to $A$. Since $\text{RREP}_{T \to D}$ does not contain new information for $A$ (a route to $D$ is already known), node $A$ does not update its routing table and, according to the specification, will not forward the RREP message to the source $T$. This leads to an unsuccessful route discovery process for node $T$.

The solution proposed by the RFC is to initiate a new route discovery process, if no route has been established 2 seconds after the first request was sent; the number of retries is flexible, but the specification recommends one retry only. In the example node $T$ would initiate another route request; node $A$, which receives the RREQ message, will immediately unicast a RREP message back to $T$.

**Local Repair.** In case of a link break, the node upstream of that break can choose to repair the link locally if the destination was no farther away than a predefined number of hops (the number is specified by the user and often depends on the network size). When a node receives a RREP message or a data packet destined for a node for which it does not have a valid route, the node buffers the message and initiates a new route discovery process. As soon as a route has been re-established, the buffered message is sent.

### 2.3 Modelling AODV and Its Variants in Uppaal

Table 1 lists all variants of AODV that are modelled, analysed and compared in this paper. The analysis is performed by SMC-Uppaal, the statistical extension

[2] A similar example has been published at the IETF mailing list in 2004; `http://www.ietf.org/mail-archive/web/manet/current/msg05702.html`.

[3] Routing tables are not presented in the figure.

**Table 1.** Different Variants of AODV

| name | optional features | remark |
|---|---|---|
| *basic* | none | follows description of Sect. 2.1 |
| *resend* | resending RREQ | “standard” configuration of AODV |
| *dflag* | D-flag | the flag is set for all route discovery processes |
| *dflag_res* | D-flag and resending | this configuration has a flaw (see below) |
| *dflag_res'* | D-flag and resending | not following the RFC literally, but flaw fixed |
| *repair* | local repair | use local repair |

of Uppaal [3]. The modelling language for SMC-Uppaal is the same as for “standard” Uppaal, namely networks of guarded, timed and probabilistic automata.

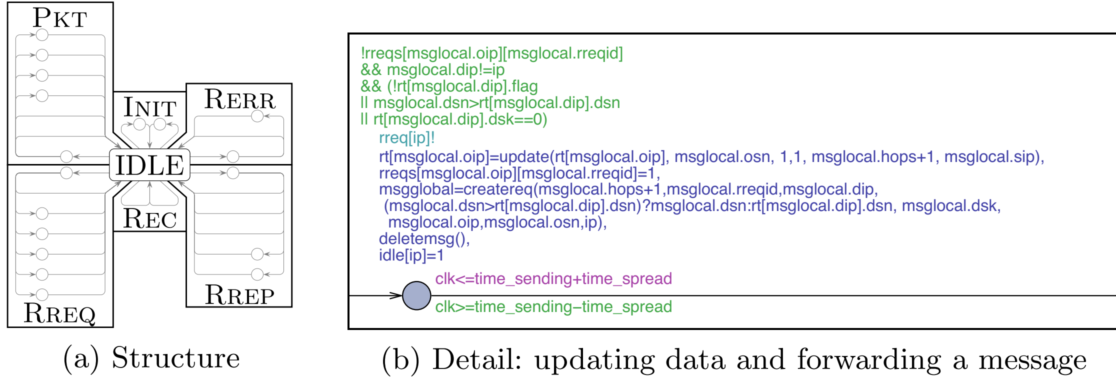
Our 6 models of (all variants of) AODV are based on a single untimed Uppaal model that was used to analyse some basic qualitative properties [7].[4] Since we are interested in a quantitative analysis of the protocol, the model had to be equipped with time and probability. The latter is needed to model dynamic topologies and mobile nodes. Hence, the (untimed) model was significantly redesigned and extended to include timing constraints on sending messages between nodes. Both the untimed and the timed model were systematically derived from an unambiguous process-algebraic model that models the intention of the RFC and does not contain contradictions. Communication between nodes had to be modelled so that the unicast behaviour of AODV was correctly rendered using SMC-Uppaal's (only) broadcast mechanism.

Each node of a network is modelled by two timed automata: the first models a message queue that buffers received messages, the other models the AODV routine. This main routine consists of $\sim 20$ locations, 1 clock measuring the sending time, and a complicated data structure with approx. 10 variables. The latter includes an array `rt` of length $N$ modelling the routing table, where $N$ is the number of nodes in the network. The overall structure of the main automaton is depicted in Fig. 2(a), it consists of 7 regions. If the automaton is in the region Idle, which consists of one location only, then AODV does not perform any action in the moment and the automaton is ready to receive messages. This happens in Rec if there is at least one message buffered. The regions Rreq, Rrep, Rerr and Pkt perform actions depending on the type of the received message. Rreq for example handles route request messages. Init initiates the transmission of data injected by the user as soon as the route is established.

Message handling often contains actions for updating the internal data (such as routing tables) and sending of a message. Fig. 2(b) gives an impression of such an update by showing a snippet of the automaton modelling the forwarding of a RREQ message.

Message sending is the only action that takes time: according to the specification of AODV [15], the most time consuming activity is the communication between nodes, which takes on average 40 milliseconds; all other times are marginal and assumed to be 0.

[4] Our models can be found at `http://www.hoefner-online.de/formats2013/`.




(a) Structure (b) Detail: updating data and forwarding a message

**Fig. 2.** Overall structure of the SMC-Uppaal model of AODV

Our models cover all core components of AODV. However, we encoded one main assumption: whenever a message is sent and the receiver of the message is within transmission range, the message will be received. In reality message loss during transmission happens regularly, for example due to communication failures or packet collisions. This loss could easily be modelled using Uppaal's broadcast mechanism in combination with probabilistic automata. However, this abstraction enables us to interpret a failure of guaranteed message delivery as an imperfection in the protocol, rather than as a result of a chosen formalism not allowing guaranteed delivery. Due to lack of space we cannot give more details about the modelling; more details about the model *basic* can be found in [11].

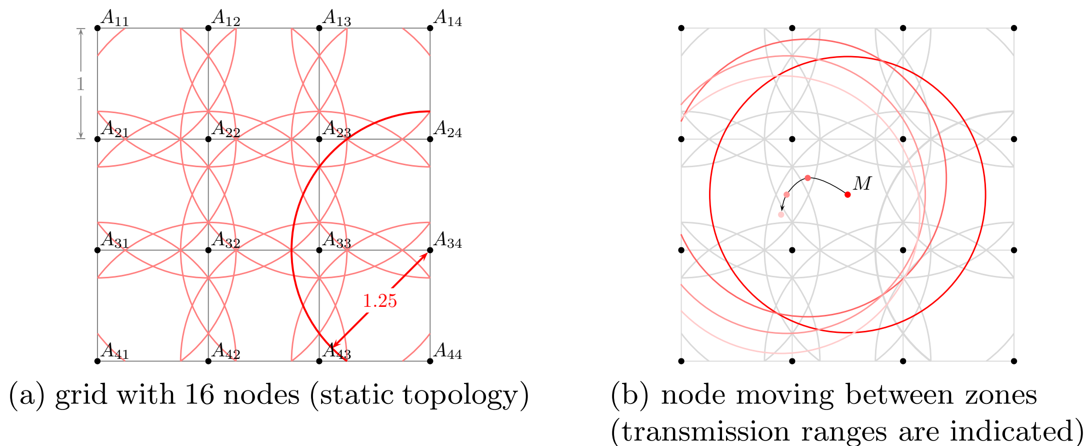
Next to the automata modelling the behaviour of the node, two additional automata are needed: the first is a scenario generator initiating the route discovery process, i.e., it forces one of the nodes to generate and broadcast a RREQ message. The second automaton models the mobility within the network.

## 3 Modelling Dynamic Topologies

To analyse quantitative properties of AODV and to compare different variants in a dynamic network, we use a *topology-based mobility model* [9]. It reflects the impact of mobility on the network topology and distinguishes static and mobile nodes; only connections to and from mobile nodes can change. Each movement is characterised either by adding a new link to or by removing an existing link from the *connectivity graph*. Whenever a mobile node $M$ enters the transmission range of a node $A$, a new link is established between $M$ and $A$. If $M$ leaves the transmission range, the link between these two nodes is removed.

To decrease the number of possible topology changes due to a large number of mobile nodes, we set up the topology as follows: the network consists of 16 static and one mobile node.[5] The static nodes form a 2-dimensional rectangular grid with grid size 1, i.e. the smallest distance between two nodes is 1 unit (cf. Fig. 3(a)); the transmission range is set to 1.25. In reality, 1 unit might correspond to 100 metres, the transmission range to 125 m, a realistic value.

[5] We also performed experiments with more than one mobile node; but these experiments do not show new (odd) results.




(a) grid with 16 nodes (static topology)

(b) node moving between zones (transmission ranges are indicated)

**Fig. 3.** Topology-based mobility model

The transmission ranges of the nodes $A_{ij}$ $(1 \leq i, j \leq 4)$ split the grid into 102 different zones. The different zones are shown in Fig. 3(a). When a mobile node $M$ moves within a zone, the exact position of the node does not matter, since it does not enter or leave the transmission range of any node—the connectivity graph stays the same. For example any node that is within the central zone is connected to nodes $A_{22}$, $A_{23}$, $A_{32}$ and $A_{33}$ (cf. Fig. 3(b)) When $M$ transits the border of a zone, it triggers a network topology change. Only the change of the connectivity graph is considered, other details such as the exact direction and angle of transmitting are not needed for characterising the dynamic network. In the example given in Fig. 3(b), $M$ moves to the left and enters the transmission range of $A_{21}$. Next, in fainter colours, the node enters transmission range of $A_{31}$ and leaves the range of $A_{23}$.

The topology-based model captures the topology changes as a Markovian transition function $\texttt{prob}(T_1, T_2)$, that assigns to two topologies $T_1$ and $T_2$ a transition probability. The probability of moving from one zone to a neighbouring zone is based on the ratio of the length the two zones share compared to the overall border length of the zone in which the node is in. For instance, the probability of transiting from the central segment of the grid to any adjacent zone is $\frac{1}{8}$, due to equal segment lengths.

Our model sets the speed of the mobile node in such a way that the node has to change zones every 35–45 time units, where the probability to leave the zone at time $t$ is equally distributed in the interval.

The zones can be grouped by their shapes; each shape forms an equivalence class. The Uppaal model reflects this observation. Each mobile node is modelled by a separate timed and probabilistic automaton; each location of the automaton characterises exactly one equivalence class. (See [9] for details.)

## 4 Experiments

Our experiments analyse the impact of mobile nodes and dynamic topologies on AODV; they are grouped into several categories: the first category analyses the

probability of route establishment for a single route discovery process, i.e., an originator node `oip` is searching for a route to `dip`; the second category analyses the likelihood of route establishment between `oip` and `dip` when additional route discovery processes occur; the last category changes the speed of the mobile node.

Before discussing the experiments, we briefly describe some foundations of statistical model checking. SMC [19,18] combines ideas of model checking and simulation with the aim of supporting quantitative analysis as well as addressing the size barrier that prevents useful analysis of large models. By trading certainty for approximation, it uses Monte Carlo style sampling, and hypothesis testing to interpret the results. The sampling follows the probability distribution defined by the non-deterministic and probabilistic automata. Parameters setting thresholds on the probability of false negatives ($\alpha$) and on probabilistic uncertainty ($\varepsilon$) can be used to specify the statistical confidence on the result. SMC-Uppaal computes the number of simulation runs needed by using Chernoff-Hoeffding bounds, which is independent of the size of the model; it generates an interval $[p-\varepsilon, p+\varepsilon]$ for estimating $p$, the probability of CTL-property $\psi$ holding w.r.t. the underlying probability distribution.

For most of our experiments we use "only" a confidence level of 95% and allow a large probabilistic interval of 10%—this is the default setting of SMC-Uppaal and means that both $\alpha$ and $\varepsilon$ are set to 5%. When using this set up, SMC-Uppaal simulates 738 runs to determine the probability of a property.

Experiments with $\alpha = \varepsilon = 1\%$ (26492 runs) are also feasible with a standard desktop machine, but require much more time. While an experiment using a confidence level of 95% takes only a couple of minutes; an experiment using a level of 99% takes more than 3 hours. We illustrate this by our first experiment.

### 4.1 Single Route Discovery Process

Our first experiment is based on 17 nodes; 16 forming a grid (Fig. 3(a)) and one mobile node $M$ which is located in the middle of the grid at the beginning of the experiment. After a delay between 140 and 160 time units (the time that the mobile node needs to perform four movements) the first RREQ message is broadcast. By this delay, the location of $M$ is random at the point the route discovery process is initiated.

In the experiment $A_{11}$ searches for a route to $A_{44}$, that means it initiates a route discovery process. We are interested whether (and at which time) $A_{11}$ establishes a route to $A_{44}$. In Uppaal syntax this reachability property is

```
Pr[<=2000](<>A11.rt[A44].nhop!=0) .        (1)
```

Checking this query determines the probability (`Pr`) satisfying the CTL-path expression `<>(A11.rt[A44].nhop!=0)` with a time bound of 2000 time units; we choose this bound as a conservative upper bound to ensure that the analyser explores paths to a depth where the protocol is guaranteed to have terminated. The term `ip.rt[dip]` refers to a route to `dip` stored inside the routing table of node `ip`. Whenever the next hop `nhop` is set ($\neq 0$), a route has been established.

The results are summarised in Table 2. From an experimental point of view, the table shows that a confidence level of 99% does not yield much better results

**Table 2.** Single Route Discovery Ratio (confidence level 95% and 99%)[6]

| | model | conf. level | probability route discovery | time route discovery | Uppaal running time |
|---|---|---|---|---|---|
| 1. | *basic* | 95% | [55.4336, 65.4336] | 595.67 | $4m\ 18s$ |
| 2. | *basic* | 99% | [59.7806, 61.7806] | 597.52 | $157m\ 44s$ |
| 3. | *resend* | 95% | [95.00, 100.00] | 847.04 | $6m\ 03s$ |
| 4. | *resend* | 99% | [98.9623, 100.00] | 836.87 | $209m\ 43s$ |
| 5. | *dflag* | 95% | [54.4851, 64.4851] | 597.50 | $4m\ 52s$ |
| 6. | *dflag* | 99% | [59.5655, 61.5655] | 597.63 | $164m\ 21s$ |
| 7. | *dflag_res* | 95% | [64.5122, 74.5122] | 698.70 | $7m\ 47s$ |
| 8. | *dflag_res* | 99% | [68.2133, 70.2133] | 688.79 | $249m\ 12s$ |
| 9. | *dflag_res'* | 95% | [81.3144, 91.3144] | 822.89 | $7m\ 08s$ |
| 10. | *dflag_res'* | 99% | [83.4104, 85.4104] | 807.43 | $230m\ 57s$ |
| 11. | *repair* | 95% | [59.7696, 69.7696] | 607.86 | $7m\ 31s$ |
| 12. | *repair* | 99% | [63.5742, 65.5742] | 606.53 | $165m\ 11s$ |

than a confidence level of 95%; but the running times of Uppaal (last column) are much higher (in average by a factor of 33.6).

Next to the running times of Uppaal the table lists the model (first column), the probability of a successful route discovery (third column) and the average time needed to establish a route between $A_{11}$ and $A_{44}$ (fourth column). It is no surprise that the models *basic* and *dflag* yield the same results—in this setting they behave identically. Furthermore, it is obvious that the probability for successful route discovery increases when using the *resend* option, while at the same time the discovery time increases as well. However, the experiments reveal three surprising and unexpected observations concerning AODV.

**Observation 1.** A single mobile node can already have a massive impact on the success of route discovery. In our setting the probability of route discovery can decrease by about 40%.

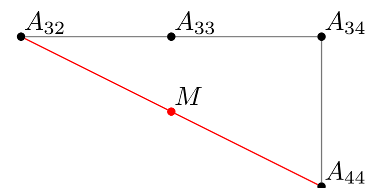



**Fig. 4.** Mobile node shortens distance

Using the same setting without mobility (e.g., the mobile node does not exist or keeps sitting in the centre of the grid), the probability of route discovery success is 100%. The success rate in our experiment using AODV *basic*, is only 60.78 ±1% (Row 2 of Table 2). The setting of the experiment guarantees that the RREQ reaches the destination $A_{44}$ and that $A_{44}$ will

[6] We use a standard computer equipped with a 3.1 GHz Intel Pentium 5 CPU, 16 GB memory, running a Mac OS operating system. As SMC-tool, we use SMC-Uppaal, the Statistical extension of Uppaal (release 4.1.11) [3], which supports both timed and probabilistic systems. Timing aspects are heavily needed to model AODV (cf. Sect. 2); the topology-based mobility model relies on probabilistic choices to determine the movement of the node.

generate a route reply. It means that the route reply, which is unicast back via a previously established path gets lost. Since the experiment consists of a single request only, RREP messages are not dropped and situations as the one sketched in Fig. 1 cannot occur. As a consequence, failure in route discovery means that a RREP message could not be unicast, which means that the established route from $A_{44}$ to $A_{11}$ uses the mobile node.

At first glance it seems to be impossible that 40% of all established routes use the mobile node as intermediate hop. But a closer analysis on time interval when a route for $A_{11}$ is discovered shows that this is in fact the case since a route via a mobile node can shorten the distance between originator and destination. In general, AODV prefers shorter routes, hence it would choose the route via the mobile node $M$. Fig. 4 illustrates how a mobile node decreases the distance between $A_{32}$ and $A_{44}$ from 3 hops to 2. The lesson learned is that static nodes should be set up in a way that it is unlikely for a mobile node to shorten the distance, or static and mobile nodes should be distinguished and routes via static nodes only should be preferred, even if they are longer.

**Observation 2.** The model *dflag_res* does not yield much improvement w.r.t. route discovery compared to *basic* and is much worse than using *resend* alone.

The chance that a route is established by the first route discovery process is around 60% (cf. *basic*). In case no route is established (chance ∼ 40%), a new request is issued; the chance that this second request yields a route establishment between $A_{11}$ and $A_{44}$ is again 60%. Putting these numbers together the success rate for *dflag_res* should be $0.6 + 0.4 \cdot 0.6 \approx 0.84 = 84\%$. Surprisingly, the probability determined by our experiments is only around 70% in case of *dflag_res* (Row 7 and 8 of Table 2). That means that many RREP messages are lost (using the same reasoning as before, no RREQ message is lost). The explanation lies in the RREP-forwarding mechanism of AODV. As explained in Sect. 2, RREP messages are not forwarded if they do not contain new information. Let us now assume that the first RREQ reaches the destination $A_{44}$, which unicasts a RREP message to the next hop on the route back to $A_{11}$, say to node $A_{34}$. This reply gets lost afterwards. Since the resend-option is set, the originator issues another request, which also reaches $A_{44}$. In case the route to $A_{11}$ is not changed in $A_{44}$'s routing table, $A_{44}$ sends another RREP message to $A_{34}$. This message does not contain new information and is dropped by the intermediate node.

To repair this flaw, we change the RREP-generation procedure. Whenever a RREP message is generated, a counter (the sequence number), which indicates the freshness of the message is incremented.[7] This change is implemented in *dflag_res'*; the evaluation results for this model are now as expected.

**Observation 3.** AODV's option of intermediate route reply should be used.

Let us have a look at the models *resend* and *dflag_res'*. The difference between the two models is that in the former model intermediate nodes are allowed to reply.

[7] In fact AODVv2 and LOADng, the successor protocols of AODV (still under development), implement exactly this variant.

Looking at the results, we notice a dramatic difference in the likelihood of route discovery. In the model *resend* the second request is followed by the generation of more than one RREP message. In fact, each node that established a route to $A_{44}$ during the first RREQ-RREP-cycle (before the reply was lost), will generate a RREP message. Due to this, route establishment is guaranteed. In contrast, there is only one RREP message for each and every request in *dflag_res'*. This observation clearly indicates that intermediate route reply is a useful feature. Interestingly, there seems to be the tendency of preferring protocols without this feature: the two successors of AODV, AODVv2 [16] and LOADng [6] follow this philosophy and set the D-flag as default—if at all, they allow intermediate route reply as an optional feature.

**Other Originator Nodes**
The first set of experiments considered a route discovery from $A_{11}$ to $A_{44}$, the largest distance a packet can travel in our set up. We expected to see the clearest results by using this distance. However, we also performed experiments with all other pairs of nodes. Fig. 5 summarises some results. It illustrates the probability of route-discovery ($y$-axis) depending on the distance between originator and destination ($x$-axis). Of course, the larger the distance between originator and destination, the smaller the chance of route establishment. Interestingly, there is a clear drop down at a distance of four nodes. It seems that from this point on resending guarantees the success. Moreover, the graph illustrates that exhaustive MC cannot help: MC is usually limited to topologies of up to 6 nodes, distances of 5 hops and more are not possible if one considers a non-linear topology.

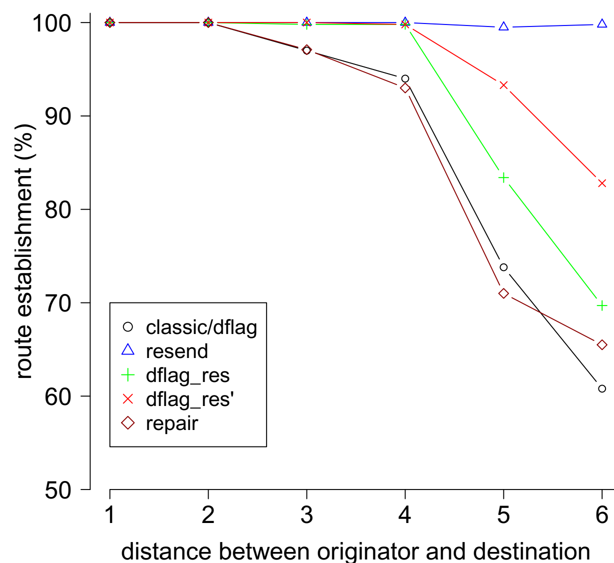



**Fig. 5.** Probability of route-establishment

## 4.2 Two Independent Route Discovery Processes

In order to evaluate the performance of variants of AODV under different network (traffic) load, we check the probability of route discovery when two route discovery processes are performed in parallel. For this second set of experiments, we are again interested in a route from $A_{11}$ to $A_{44}$. However, shortly (35-45 milliseconds) after the packet is handed over to $A_{11}$, a second data packet is injected at another node, destined for some destination; in fact we did experiments for all destinations, but present only two observations—due to lack of space.

**Observation 4.** RREP messages are dropped more often than expected.

**Table 3.** Two route discovery processes looking for the same destination $A_{44}$

| **distance between orig.** | **orginator of $2^{nd}$ request** | **class** | **probability (avg.)** |
|---|---|---|---|
| 1 | $\{A_{12}, A_{21}\}$ | nodes at border | 43.36% |
| 2 | $\{A_{13}, A_{31}\}$ | nodes at border | 17, 75% |
| | $\{A_{22}\}$ | inner node | 13.28% |
| 3 | $\{A_{14}, A_{41}\}$ | nodes at border | 43, 10% |
| | $\{A_{23}, A_{32}\}$ | inner nodes | 29.74% |
| 4 | $\{A_{24}, A_{42}\}$ | nodes at border | 71, 61% |
| | $\{A_{33}\}$ | inner node | 47.29% |
| 5 | $\{A_{34}, A_{43}\}$ | nodes at border | 80.42% |

We consider the scenario where the second request is sent from $A_{22}$ to $A_{44}$. Since some nodes drop RREP messages (cf. Sect. 2.2), the probability of route establishment between $A_{11}$ and $A_{44}$ should decrease (compared to the 60% of Table 2). However, SMC-Uppaal shows that the probability of $A_{11}$ finding a route to $A_{44}$ is in the probability interval $[8.27913, 18.2791]$, i.e., a route discovery is unlikely. More results for the *basic* model are summarised in Table 3, grouped by the distance between the two originators. The table lists only the originator of the second route request; both the originator ($A_{11}$) of the first request and the destination ($A_{44}$) of both requests are fixed.

There is a correspondence between the success of route discovery and the distance between the two originators; moreover inner nodes have more influence on route discovery than nodes lying on the border of the network. This shows that the example of Fig. 1 occurs regularly. However, if the second originator $\mathtt{oip}_2$ is far away from the first originator $A_{11}$ no RREP message is dropped, since a route between $\mathtt{oip}_2$ and $A_{44}$ is established before the RREQ from $A_{11}$ reaches $\mathtt{oip}_2$. In the case of $\mathtt{oip}_2 \in \{A_{24}, A_{42}, A_{34}, A_{43}\}$, the probability even increases. This is in line with Observation 3: when intermediate route reply is enabled, more RREP message are generated and the probability of route discovery success grows.

**Observation 5.** "Busy" mobile nodes increase the chance of route discovery.

One could rephrase this observation to "adding noise sometimes increases performance". At first glance it seems that adding additional network traffic—here a second route discovery processes—should not increase performance. But, let us look the scenario where the first data packet needs to be send from $A_{11}$ to $A_{44}$ (as before); the second packet is sent from $A_{31}$ to the mobile node $M$. While handling the second RREQ most of the nodes will not learn about $A_{44}$ and $A_{11}$. However it turns out that in the *basic* model, the probability of route discovery increases from around 60% to 72%. One reason is that the mobile node handles the request and generates a RREP message. While doing this it cannot handle the first RREQ; in case the first RREQ is sent to $M$ and it is handling a different messages (is busy), the message is buffered. If the message is buffered for a while, the chance that the RREQ from $A_{11}$ reaches $A_{44}$ via a path without $M$ as an

**Table 4.** different mobile node speed and impact on AODV variants

| **model** | **probability**$_{\text{fast}}$ | **probability**$_{\text{moderate}}$ | **probability**$_{\text{slow}}$ |
|---|---|---|---|
| *basic* | [48.5230, 58.5230] | [55.4336, 65.4336] | [61.9377, 71.9377] |
| *resend* | [94.8645, 100.00] | [95.00, 100.00] | [94.3225, 100.00] |
| *dflag* | [50.0136, 60.0136] | [54.4851, 64.4851] | [63.2927, 73.2927] |
| *dflag_res* | [60.1762, 70.1762] | [64.5122, 74.5122] | [70.6098, 80.6098] |
| *dflag_res'* | [75.8943, 85.8943] | [81.3144, 91.3144] | [85.5149, 95.5149] |
| *repair* | [54.0786, 64.0786] | [57.6016, 67.6016] | [65.5962, 75.5962] |

intermediate hop, increases. Hence not the shortest, but the “fastest” route is established from $A_{44}$ to $A_{11}$; this route is then used to send the RREP, since it does not use the mobile node as intermediate hop, the RREP is not lost.

### 4.3 Influence of Speed of Mobile Nodes

In our experiments the topology changes within a time frame of 35 to 45 milliseconds; This also determines the speed of the mobile node. One might argue that the speed of the mobile node affects our analysis and that other speeds could yield different behaviour. As shown in Table 4, this is not the case—the probabilities slightly change, but stay in the same ball park. Moreover the relationship between the different variants stays the same; a variant that is more reliable with a fast mobile node, is also more reliable with a slower node. For this category of experiments we enforce a topology change within the interval $[25, 35]$ (fast), $[35, 45]$ (moderate), and $[95, 105]$ (slow), respectively.

## 5 Related Work

Model checking has been used to analyse routing protocols in general and AODV in particular. For example, Bhargavan et al. [1] were amongst the first to use model checking—they used the SPIN model checker—on a draft of AODV, demonstrating the feasibility and value of automated verification of routing protocols. Musuvathi et al. [14] introduced the CMC model checker primarily to search for coding errors in implementations of protocols written in C. They used AODV as an example and, as well as discovering a number of errors, they also found a problem with the specification itself, which has since been corrected. Chiyangwa and Kwiatkowska [4] used the timing features of UPPAAL to study the relationship between the timing parameters and the performance of route discovery. None of these studies performed a quantitative analysis of AODV.

Statistical model checking techniques [19,18] are rather new. So far they have been used in a couple of case studies. Bulychey et al. [2] for example apply the SMC-Uppaal to an analysis of an instance of the Lightweight Media access

Control (LMAC) protocol; by this they are able to analyse ring topologies of up to 10 nodes.[8] Applications of SMC within biological systems are discussed in [5,13]. To the best of our knowledge, SMC was not used for the analysis of routing protocols so far—except in [11], where SMC-Uppaal is used to compare AODV and DYMO and to illustrate that even large topologies (up to 100 nodes) can be analysed by SMC. Our experiments are in line with this. However, it is unique in the sense that we carefully study variants of AODV.

## 6 Conclusion and Future Work

The aim of this paper has been a careful (quantitative) analysis of AODV and its variants using statistical model checking techniques. By this, we have made surprising observations on the behaviour of AODV. We have shown for example that some optional features (D-flag) should not be combined with others (resending). Another result shows that a well-known shortcoming occurs more often than expected and has a tremendous effect on the success of route establishment. One challenge we faced while performing our experiments has been the interpreting the data.

The results were often surprising and hard to interpret, particularly when they indicate odd behaviour. Unfortunately SMC-Uppaal does not store traces during analysis, thus it is difficult to recover counterexamples to explain the observations. At the moment counter examples are constructed "by hand" by formulating more probing queries beyond looking at overall performance. This suggests that more powerful statistical analysis such as "rare event simulation" in combination with multiple queries could be used to compile better evidence.

Next to this careful analysis, we also showed that SMC is a suitable tool for analysing WMNs. In this setting classical MC was limited to topologies with up to 6 nodes and therefore having a realistic mobility model was not possible.

Future work will be a continuation of our case study. In particular we want to look at topologies of up to 100 nodes—it has been shown that an analysis of such networks is possible [11]. However, choosing the right scenario is crucial here: Since one cannot analyse all scenarios, one has to pick the right topologies and the right mobility model(s); but in some sense finding the correct setting becomes a "stab in the dark". We hope that our previous experience helps to set the experiments right.

**Acknowledgement.** We thank Ansgar Fehnker, Rob van Glabbeek and Franck Cassez for fruitful discussions and their help.

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

[8] Other case studies include firewire, bluetooth, and a train gate (see `http://people.cs.aau.dk/~adavid/smc/cases.html` for an overview).

## References


1. Bhargavan, K., Obradovic, D., Gunter, C.A.: Formal verification of standards for distance vector routing protocols. J. ACM 49(4), 538–576 (2002)
2. Bulychev, P., David, A., Guldstrand Larsen, K., Legay, A., Mikučionis, M., Bøgsted Poulsen, D.: Checking and distributing statistical model checking. In: Goodloe, A.E., Person, S. (eds.) NFM 2012. LNCS, vol. 7226, pp. 449–463. Springer, Heidelberg (2012)
3. Bulychev, P., David, A., Larsen, K., Mikučionis, M., Bøgsted Poulson, D., Legay, A., Wang, Z.: UPPAAL-SMC: Statistical model checking for priced timed automata. In: Wiklicky, H., Massink, M. (eds.) Quantitative Aspects of Programming Languages. EPTCS, vol. 85, pp. 1–16. Open Publishing Association (2012)
4. Chiyangwa, S., Kwiatkowska, M.: A timing analysis of AODV. In: Steffen, M., Zavattaro, G. (eds.) FMOODS 2005. LNCS, vol. 3535, pp. 306–321. Springer, Heidelberg (2005)
5. Clarke, E.M., Faeder, J.R., Langmead, C.J., Harris, L.A., Jha, S.K., Legay, A.: Statistical Model Checking in BioLab: Applications to the automated analysis of T-cell receptor signaling pathway. In: Heiner, M., Uhrmacher, A.M. (eds.) CMSB 2008. LNCS (LNBI), vol. 5307, pp. 231–250. Springer, Heidelberg (2008)
6. Clausen, T., Colin de Verdiére, A., Yi, J., Niktash, A., Igarashi, Y., Satoh, H., Herberg, U., Lavenu, C., Lys, T., Perkins, C., Dean, J.: The lightweight on-demand ad hoc distance-vector routing protocol - next generation (LOADng). Internet Draft (Standards Track) (2013), `http://tools.ietf.org/html/draft-clausen-lln-loadng-08`
7. Fehnker, A., van Glabbeek, R., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: Automated analysis of AODV using UPPAAL. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 173–187. Springer, Heidelberg (2012)
8. Fehnker, A., van Glabbeek, R.J., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV. Tech. Rep. 5513, NICTA (2013), `http://www.nicta.com.au/pub?id=5513`
9. Fehnker, A., Höfner, P., Kamali, M., Mehta, V.: Topology-based mobility models for wireless networks. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 389–404. Springer, Heidelberg (2013)
10. Höfner, P., van Glabbeek, R., Tan, W., Portmann, M., McIver, A., Fehnker, A.: A rigorous analysis of AODV and its variants. In: Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWIM 2012, pp. 203–212. ACM (2012)
11. Höfner, P., McIver, A.: Statistical model checking of wireless mesh routing protocols. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 322–336. Springer, Heidelberg (2013)
12. IEEE P802.11s: IEEE draft standard for information technology—telecommunications and information exchange between systems—local and metropolitan area networks—specific requirements—part 11: Wireless LAN Medium Access Control (MAC) and physical layer (PHY) specifications-amendment 10: Mesh networking (July 2010)
13. Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A bayesian approach to model checking biological systems. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 218–234. Springer, Heidelberg (2009)
14. Musuvathi, M., Park, D.Y.W., Chou, A., Engler, D.R., Dill, D.L.: CMC: a pragmatic approach to model checking real code. In: Operating Systems Design and Implementation, OSDI 2002 (2002)

15. Perkins, C., Belding-Royer, E., Das, S.: Ad hoc on-demand distance vector (AODV) routing. RFC 3561 (Experimental) (2003), http://www.ietf.org/rfc/rfc3561
16. Perkins, C., Chakeres, I.: Dynamic MANET on-demand (AODVv2) routing. Internet Draft (Standards Track) (2013), http://tools.ietf.org/html/draft-ietf-manet-dymo-25
17. Perkins, C., Royer, E.: Ad-hoc On-Demand Distance Vector Routing. In: 2nd IEEE Workshop on Mobile Computing Systems and Applications, pp. 90–100 (1999)
18. Sen, K., Viswanathan, M., Agha, G.A.: Vesta: A statistical model-checker and analyzer for probabilistic systems. In: Quantitative Evaluaiton of Systems, QEST 2005, pp. 251–252. IEEE (2005)
19. Younes, H.: Verification and Planning for Stochastic Processes with Asynchronous Events. Ph.D. thesis, Carnegie Mellon University (2004)

# Paper V

## Refinement-Based Modeling of 3D NoCs

**Maryam Kamali, Luigia Petre, Kaisa Sere and Masoud Daneshtalab**

Originally published in: F. Arbab and M. Sirjani (Eds.), *Proceedings of the 4th IPM International Conference on Fundamentals of Software Engineering - FSEN 11*, Lecture Notes in Computer Science Vol. 7141, pp. 236-252, Springer-Verlag, 2012.

# Refinement-Based Modeling of 3D NoCs

Maryam Kamali[1,2], Luigia Petre[1], Kaisa Sere[1], and Masoud Daneshtalab[3]

[1] Åbo Akademi University, Finland
[2] Turku Centre for Computer Science (TUCS), Finland
[3] University of Turku, Finland

**Abstract.** Three-dimensional Networks-on-Chip (3D NoC) have recently emerged essentially via the stacking of multiple layers of two-dimensional NoCs. The resulting structures can support a very high level of parallelism for both communication and computation as well as higher speeds, at the cost of increased complexity. To address the potential problems due to the highly complex NoCs, we study them with formal methods. In particular, we base our study on the *refinement* relation between models of the same system. We propose three abstract models of 3D NoCs, $M_0$, $M_1$, and $M_2$ so that $M_0 \sqsubseteq M_1 \sqsubseteq M_2$, where '$\sqsubseteq$' denotes the refinement relation. Each of these models provides templates for communication constraints and guarantees the communication correctness. We then show how to employ one of these models for reasoning about the communication correctness of the XYZ-routing algorithm.

## 1 Introduction

The Network-on-Chip (NoC) architecture paradigm, based on a modular packet-switching mechanism, can address many of the on-chip communication design issues such as performance limitations of long interconnects and the integration of high numbers of Intellectual Property (IP) cores on a chip. However, the 2D-chip fabrication technology faces many challenges in the deep submicron regime even when employing NoC architectures, e.g, the design of the clock-tree network for large chips, limited floor-planning choices, the increase of both the wire delay and power consumption, the integration of various components that are digital, analog, MEMS and RF, etc. Three Dimensional Integrated Circuits (3D ICs) have been emerging as a viable candidate to achieve better performance and package density as compared to traditional Two Dimensional (2D) ICs. In addition, combining the benefits of 3D ICs and NoC schemes provides a significant performance gain for 3D architectures [13,27,22].

Three dimensional Networks-on-Chip (3D NoCs) [13] provide more reliable interconnections due to the increased number of links between components. Due to their promise of parallelism and efficiency, 3D NoCs have a critical role in leading towards reliable computing platforms. However, the majority of their evaluation approaches are simulation-based tools, such as XMulator [25], Noxim [26], etc. Simulation-based approaches are usually applied in the late stages of design and are limited, e.g., by the length of time that a system is simulated.

F. Arbab and M. Sirjani (Eds.): FSEN 2011, LNCS 7141, pp. 236–252, 2012.
© Springer-Verlag Berlin Heidelberg 2012

This means that exhaustive checking of all the system states is impossible in practice for complex 3D NoCs and thus, simulation is not suitable for verifying the correctness of a NoC design.

Another approach to address this problem is via formal methods. Formal methods refer to the application of mathematical techniques to the design and implementation of computer hardware and software. Prominent examples of applying formal methods are provided by, e.g., Intel [16,19] and IBM [21] for formally verifying hardware or systems-on-chip (SoC) [15]. By using rigorous mathematical techniques, it is possible to deliver *provably correct* systems. Formal methods are based on the capture of system requirements in a specific, precise format. Importantly, such a format can be analyzed for various properties and, if the formal method permits, also stepwise developed until an implementation is formed. By following such a formal development, we are *sure* that the final result correctly implements the requirements of the system.

Much of the research concerning the 3D NoC design is concentrated on various bottom-up approaches, such as the study of routing algorithms [6,20] or the design of dedicated 3D NoC architectures [29] where parameters such as hop count or power consumption are improved. Here we are concerned with a reverse, top-down approach where we start from simple models and add complexity later. There are already research results regarding the detection of faults as well as debugging in the early stages of NoC design. A generic model for specifying and verifying NoC systems is presented in [10] where the formal verification is addressed with the ACL2 theorem prover, a mechanized proof tool. This tool produces a set of proof obligations that should be discharged for particular NoC instances. This generic model has been used for verification of functionality features in 2D-NoC systems. Another formal approach to the development of the NoC systems employing the B-action systems formalism has been described in [28], where the focus is on the formal specification of communication routers. A framework for modeling 2D-NoC systems by composing more advanced routing components out of simpler ones, is proposed there.

In this paper, we go one step further and propose a top-down formalization of the early *3D* NoC design. The formal method we employ is Event-B [2] which comes with the associated tool Rodin [1,30]. One of the main features of Event-B is that the system development is done in a stepwise manner that eventually leads to a system implementation. The stepwise development is captured by the *refinement* [4,5] relation between models of the same system, so that a high-level model of a system is transformed by a sequence of correctness-preserving steps into a more detailed and efficient model that satisfies the original specification. We specify here the general structure of a 3D NoC at a high level of abstraction in Event-B. The specification formulates the main constraints of the communication model, needed to prove its correctness. Our definition for correctness at this abstract level of modeling is to show that a package injected in the network is eventually received at the destination. We propose three different abstract models $M_0$, $M_1$, and $M_2$ for a 3D NoC so that $M_0 \sqsubseteq M_1 \sqsubseteq M_2$, where '$\sqsubseteq$' denotes the refinement relation. Furthermore, each of these models can be refined

into more concrete models to define specific 3D NoC designs in the early stages of the system development. When the concrete models preserve the correctness properties of the abstract models, we guarantee the correctness of the concrete 3D NoC designs. As an application of the general 3D NoC designs, we model the XYZ routing algorithm by refining the $M_2$ abstract model. To verify the XYZ routing algorithm, we generate the proof obligations using the Rodin tool and discharge them automatically or interactively.

We proceed as follows. In Section 2 we overview the Event-B formal method to the extent needed in this paper. In Section 3 we propose three increasingly more detailed formal models for a 3D NoC together with the constraints for proving correctness. In Section 4 we illustrate the formal modeling of the XYZ routing algorithm as a case study. In Section 5 we discuss the proof obligations while in Section 6 we present concluding remarks and future work.

## 2 Preliminaries

Event-B [2,1] is an extension of the B formalism [3,28] for specifying distributed and reactive systems. A system model is gradually specified on increasing levels of abstraction, always ensuring that a more concrete model is a *correct implementation* of an abstract model. The language and proof theory of Event-B are based on logic and set theory. The correctness of the stepwise construction of formal models is ensured by discharging a set of proof obligations: if these obligations hold, then the development is mathematically shown to be correct. Event-B comes with the associated tool Rodin [1,30], which automatically discharges part of the proof obligations and also provides the means for the user to discharge interactively the remaining proofs.

Each Event-B model consists of two components called *context* and *machine*. A context describes the static part of the model, i.e., it introduces new types and constants. The properties of these types and constants are gathered as a list of axioms. A machine represents the dynamic part of the model, consisting of variables that define the *state* of the model and operations called *events*. The system properties that should be preserved during the execution are formulated as a list of *invariant* predicates over the state of the model.

An event, modeling state changes, is composed of a *guard* and an *action*. The guard is the necessary condition under which an event might occur; if the guard holds, we call the event *enabled*. The action determines the way in which the state variables change when the event occurs. For initializing the system, a sequence of actions is defined. When the guards of several events hold at the same time, then only one event is non-deterministically chosen for execution. If some events have no variables in common and are enabled at the same time, then they can be considered to be executed in parallel since their sequential execution in any order gives the same result. For all practical purposes, this execution model is parallel and can be implemented as such when the model is refined to code. Events can be declared as *anticipated*, meaning that in the future refinements we need to set out a natural number expression called *variant* and prove that it

is decreased by this event. Events can also be *convergent*, meaning that in the current machine there is a variant that decreases when this event is chosen for execution. Thus, an anticipated event is not convergent in the current machine but should become so in a future refinement of that machine.

A model can be developed by a number of correctness preserving steps called *refinements* [4,5]. One form of model refinement can add new data and new events on top of the already existing data and behavior but in such a way that the introduced behavior does not contradict or take over the abstract machine behavior. This form of stepwise construction is referred to as *superposition* refinement [18,9]. We may also use other refinement forms, e.g., *algorithmic* refinement [8]. In this case, an event of an abstract machine can be refined by several corresponding events in a refined machine. This will model different branches of execution, that can, for instance, take place in parallel and thus can improve the algorithmic efficiency. In this paper, we use only superposition refinement.

## 3 Three Abstract Models for the 3D NoC: $M_0$, $M_1$, $M_2$

In this section we formally develop three high-level models $M_0$, $M_1$, and $M_2$ for the 3D NoC. Our models are at three increasing levels of detail so that each model is a refinement of the previous one: $M_0 \sqsubseteq M_1 \sqsubseteq M_2$. In the initial model, we specify a network of nodes and define the correctness properties of this network based on a specific data structure called *pool*, as suggested by [2]. In the second model, we add new data and events to model the 3D mesh-based NoC architecture; besides, we specify the channels between nodes. In the third model, we model buffers for nodes and refine the communication model.

By starting from an initial model that is rather abstract, i.e., without detailing the communication topology, we obtain a rather general starting point that can later be refined to various topologies. Moreover, adding channels and ports only in the second model leads to a clean modelling of the basic communication mechanism (via routing and switching) in the initial model; the required detail (of channels and ports) are not needed for understanding the communication mechanism. Adding buffers in the third model illustrates an extra level of detail. Networks where the nodes have no buffers for communication will, therefore, employ the second model as their abstraction and not the third.

### 3.1 The Initial Model $M_0$

The first model $M_0$ that we construct is rather abstract: we do not consider the numerous parts of the network such as channels or buffers; they will be introduced in subsequent refinements. $M_0$ will thus allow us to reason about the system very abstractly [2]. The model $M_0$ is formed of the static part and the dynamic part, as follows.

**The Static Part.** The static part of our model is described in Fig. 1 and contains the sets *MESSAGES*, *ROUTER*, *DATA* and the constants *data*, *des*, *src* and *Neigh*. The message identifiers are modeled by the non-empty and finite

*MESSAGES* set. We use the following modeling idea for messages. A message id in the *MESSAGES* set relates to a triple *(data, source, destination)* where *data* is an element of the *DATA* set, *source* models the source node where a message is injected, and *destination* models the destination node where a message should be received. A message should not be destined to its source node. The set of network nodes and data are modeled by the sets *ROUTER* (finite and non-empty) and *DATA* (finite and non-empty), respectively. The relation *Neigh* (non-empty, symmetric, and non-reflexive) models the neighbor structure i.e., which node can communicate with which node.

**SETS** MESSAGES ROUTER DATA
**CONSTANTS** data des src Neigh
**AXIOMS**

$$MESSAGES \neq \varnothing \wedge finite(MESSAGES)$$
$$ROUTER \neq \varnothing \wedge finite(ROUTER)$$
$$DATA \neq \varnothing \wedge finite(DATA)$$
$$data \in MESSAGES \rightarrow DATA$$
$$src \in MESSAGES \rightarrow ROUTER \wedge des \in MESSAGES \rightarrow ROUTER$$
$$\forall m, sp, dp \cdot m \in MESSAGES \wedge sp \in ROUTER \wedge dp \in ROUTER \wedge m \mapsto sp \in src \wedge m \mapsto dp \in des \Rightarrow sp \neq dp$$
$$Neigh \in ROUTER \leftrightarrow ROUTER$$
$$Neigh \neq \varnothing \wedge Neigh = Neigh^{-1} \wedge dom(Neigh) \lhd id \cap Neigh = \varnothing$$

**Fig. 1.** $M_0$: the static part

To define structure types such as records in Event-B, we use functions to represent attributes. Therefore, our modeling idea translates to the functions *data*, *src* and *des* with ranges *DATA*, *ROUTER*, and *ROUTER*, respectively.

**The Dynamic Part.** In our network model we use the following condition for modeling the communication correctness: the messages in the network will eventually reach their destinations. For this, we define two message subsets and one partial message-to-node map as machine variables: $sent_pool \subseteq MESSAGES$, $received_pool \subseteq MESSAGES$ and $moving_pool \in sent_pool \nrightarrow ROUTER$.

The *sent_pool* subset denotes the list of messages injected into the network. The *sent_pool* subset is updated whenever a new message is injected into the network, while the *moving_pool* subset denotes the current position of traveling messages. All the messages injected into the network are added to the *moving_pool* and whenever a message is routed from a node to another one, the current position of that message is updated in the *moving_pool*. The *received_pool* subset denotes the list of messages received from the network by destination nodes. Whenever a message is received at its destination, it will be added to *received_pool* and removed from *moving_pool*. The behavior of message pools is illustrated in Fig. 2.

To model the communication and the message pool functions, we define three events as explained below. The *sent_message* event described in Fig. 3(a) handles the injection of a new message into the network. Whenever a message is injected into the network both *sent_pool* as well as *moving_pool* are updated.

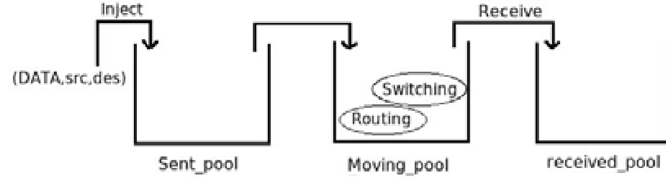



**Fig. 2.** Message Pools

```
Event sent_message ≙
    any
        current_msg
    where
        current_msg ∈ MESSAGES
        current_msg ∉ sent_pool
    then
        sent_pool := sent_pool ∪ {current_msg}
        moving_pool := moving_pool
          ∪ {current_msg ↦ src(current_msg)}
    end
```

(a) Message Injection

```
Event routing ≙
    begin
        skip
    end
Event switching ≙
Status anticipated
    any
        current_msg  new_position
    where
        current_msg ∈ dom(moving_pool)
        des(current_msg) ≠
          moving_pool(current_msg)
        new_position ↦ moving_pool(current_msg)
          ∈ Neigh
        new_position ≠ src(current_msg)
    then
        moving_pool(current_msg) := new_position
    end
```

(b) Routing and Switching

**Fig. 3.** $M_0$ Events

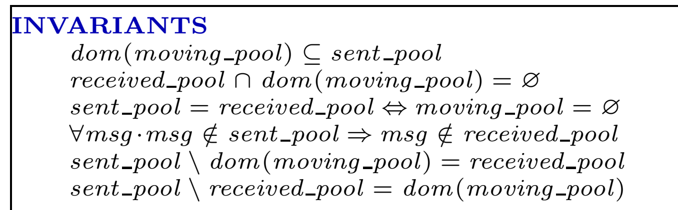
A message in *moving_pool* should be routed toward its destination. This is composed of two actions, one for deciding which node would be the next one (routing) and the other for transferring the message to that node (switching). These two actions are available for all the nodes, including the source, the destination as well as all the intermediate nodes and are modeled respectively by the *routing* and *switching* events shown in Fig. 3(b). In this abstract model we do not have any routing decisions, hence, the *routing* event is modeled by *skip*. The *switching* event in the $M_0$ model only transfers a message from the current node to one of its neighbors nondeterministically and updates the *moving_pool* by changing the current position of a message. To avoid cycling, we do not allow a message to return to its source. The reason for not considering a specific routing algorithm is that it makes our initial model more general and reusable for a wide variety of routing algorithms implementations. The *switching* event has the status *anticipated*.

```
Event received_message ≙
Status convergent
    any
        current_msg
    where
        current_msg ∈ dom(moving_pool)
        des(current_msg) = moving_pool(current_msg)
    then
        moving_pool := {current_msg} ⩤ moving_pool
        received_pool := received_pool ∪ {current_msg}
    end
```

**Fig. 4.** $M_0$: Received_message Event

The *received_message* event shown in Fig. 4 adds a message received at its destination to *received_pool* and removes the message from *moving_pool*. This event is convergent: if new messages are not injected to the network for a certain time, all the messages will be received at their destinations. This is proved by means of the ($sent_pool \setminus received_pool$) variant denoting the difference between the sets *sent_pool* and *received_pool*.

In order to prove the communication correctness, we need to prove that the *sent_pool* subset eventually becomes equal with the *received_pool* subset and the *moving_pool* subset is empty when all the messages are received at their destinations. These properties are formulated in Fig. 5 as invariants.



**INVARIANTS**

$dom(moving_pool) \subseteq sent_pool$
$received_pool \cap dom(moving_pool) = \varnothing$
$sent_pool = received_pool \Leftrightarrow moving_pool = \varnothing$
$\forall msg \cdot msg \notin sent_pool \Rightarrow msg \notin received_pool$
$sent_pool \setminus dom(moving_pool) = received_pool$
$sent_pool \setminus received_pool = dom(moving_pool)$

**Fig. 5.** $M_0$: Invariants (Pool Modeling)

$M_0$ is a general specification of a general network and will be refined to model 3D NoC communication designs in the following. Moreover, the model provides the necessary properties that should be preserved by refinement. These properties, that guarantee the overall communication correctness, are defined as the list of invariants.

### 3.2 The Second Model $M_1$

Transferring a message from a node to its neighbor in the model $M_0$ is achieved simply by copying the message from a node to another. In this section we refine the initial model $M_0$ to also specify channels specific to the 3D NoCs. To specify channels, we need a 3D NoC architecture. There are a number of 3D NoC architectures, e.g., mesh-based [13], tree-based [14]. We consider here NoCs with 3D mesh topologies. The 3D mesh-based NoC (Fig. 6(a)) consists of $N = m * n * k$ nodes; each node has an associated integer *coordinate triple* $(x, y, z)$, $0 < x \leq m$, $0 < y \leq n$, $0 < z \leq k$.

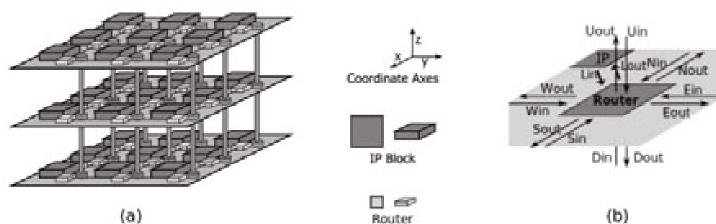



**Fig. 6.** (a) 3D Mesh-based NoC architecture (b) Router channels

Our 3D NoC architecture employs seven-port routers: one port to the IP block, one port to above and below routers, and one in each cardinal direction (North, South, East and West), as shown in Fig. 6(b).

**The Static Part.** We extend the static part of the initial model $M_0$ in three ways: we map routers to coordinate triples, we add new properties for the *neigh* relation based on the coordinate triples, and we model ports and channels for the 3D NoC. In order to map routers to the coordinate triples, we define four constants: *coordX*, *coordY*, *coordZ* and *mk_position* as shown in Fig. 7. The *coordX*, *coordY* and *coordZ* constants represent coordinate triples $(x, y, z)$ and the *mk_position* constant is a map associating each router to a position in space given by the coordinates. The *crossbarX*, *crossbarY* and *crossbarZ* constants model the number of nodes in X, Y and Z coordinate in the network, respectively.

Two nodes with coordinates $(x_i, y_i, z_i)$ and $(x_j, y_j, z_j)$ are connected by a communication channel if and only if $|x_i - x_j| + |y_i - y_j| + |z_i - z_j| = 1$. To model this neighbor structure, the $Neigh$ relation in the initial model $M_0$ is restricted in this model by adding the axiom in Fig. 8.

***SETS*** *CHANNEL PORTS*
***CONSTANTS*** *coordX coordY coordZ mk_position*
*crossbarX crossbarY crossbarZ mk_channel*
**AXIOMS**
$crossbarX \in \mathbb{N}_1 \wedge crossbarY \in \mathbb{N}_1 \wedge crossbarZ \in \mathbb{N}_1$
$mk_position \in (1 \,..\, crossbarX) \times (1 \,..\, crossbarY) \times (1 \,..\, crossbarZ) \rightarrowtail\!\!\!\!\twoheadrightarrow ROUTER$
$coordX \in ROUTER \twoheadrightarrow (1 \,..\, crossbarX)$
$coordY \in ROUTER \twoheadrightarrow (1 \,..\, crossbarY)$
$coordZ \in ROUTER \twoheadrightarrow (1 \,..\, crossbarZ)$
$\forall xx, yy, zz \cdot xx \in 1 \,..\, crossbarX \wedge yy \in 1 \,..\, crossbarY \wedge zz \in 1 \,..\, crossbarZ$
$\Rightarrow coordX(mk_position(xx \mapsto yy \mapsto zz)) = xx$
$\wedge coordY(mk_position(xx \mapsto yy \mapsto zz)) = yy$
$\wedge coordZ(mk_position(xx \mapsto yy \mapsto zz)) = zz$
$\forall pos1, pos2 \cdot pos1 \in ROUTER \wedge pos2 \in ROUTER \wedge pos1 \neq pos2$
$\Rightarrow coordX(pos1) \neq coordX(pos2) \vee coordY(pos1) \neq coordY(pos2) \vee$
$coordZ(pos1) \neq coordZ(pos2)$

**Fig. 7.** $M_1$: Static Part 1

We define the $CHANNEL$ set to model the communication channels between routers and we define the $PORTS$ set to define the input and output ports of nodes in the static part of the second model. To show how two neighbors are connected to each other through channels, we define the *def_channel* and *mk_channel* relations with the help of axioms, as shown in Fig. 8. The *def_channel* relation models the relation of a port of a node to the corresponding port of its neighbor and the *mk_channel* relation maps the port relations to channels.

**AXIOMS**
$\forall r1, r2 \cdot r1 \mapsto r2 \in Neigh \Leftrightarrow abs(coordX(r1) - coordX(r2)) + abs(coordY(r1)$
$- coordY(r2)) + abs(coordZ(r1) - coordZ(r2)) = 1$
$def_channel \in (ROUTER \times PORTS) \rightarrow (ROUTER \times PORTS)$
$partition(PORTS, \{Ein\}, \{Eout\}, \{Win\}, \{Wout\}, \{Nin\}, \{Nout\}, \{Sin\}$
$, \{Sout\}, \{Uin\}, \{Uout\}, \{Din\}, \{Dout\}, \{Lin\}, \{Lout\})$
$mk_channel \in def_channel \rightarrowtail\!\!\!\!\twoheadrightarrow CHANNELS$

**Fig. 8.** $M_1$: Static Part 2

East and west ports of neighbor nodes with different X coordinate are related to each other through a channel. For instance, as shown in Fig. 9, $Ein$ and $Eout$ ports of node $(1,1,1)$ are connected to $Wout$ and $Win$ ports of node $(2,1,1)$ through a channel $((1,1,1) \mapsto Eout) \mapsto ((2,1,1) \mapsto Win)$ and $((2,1,1) \mapsto Wout) \mapsto ((1,1,1) \mapsto Ein)$ relations in $def_channel$. This connection of the ports of the neighboring nodes on the X coordinate is modeled by the axiom shown in Fig. 10. The port relation between neighbors on other coordinates is defined by similar axioms which are not shown here due to lack of space.

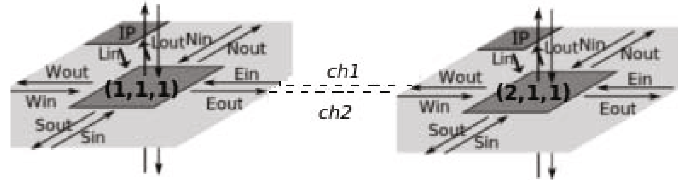



**Fig. 9.** Channels in 3D Mesh-Based NoCs

**The Dynamic Part.** In the static part of the model $M_1$, we define the 3D mesh NoC architecture with the triple coordinate of nodes and their channels. In the dynamic part of the model $M_1$, we refine the dynamic part of the model $M_0$ to specify the transferring of data through the communication channels, so that the overall correctness of communication holds.

The communication channels between routers are considered asynchronous channels, transferring data upon request. Each channel propagates data as well as control values. In our case, a control value models the fact that a channel is occupied by a message. When a message is injected to a channel, the control value of that channel is set to *busy* and when the message is received at the other side of channel, the control value of that channel is set to *free*.

**AXIOMS**

$\forall n, m, i, j \cdot (n \mapsto i) \mapsto (m \mapsto j) \in def_channel \land i = Wout \land j = Ein$
$\Leftrightarrow coordX(n) - coordX(m) = 1 \land coordY(n) = coordY(m) \land coordZ(n) = coordZ(m)$
$\forall n, m, i, j \cdot (n \mapsto i) \mapsto (m \mapsto j) \in def_channel \land i = Eout \land j = Win$
$\Leftrightarrow coordX(n) - coordX(m) = -1 \land coordY(n) = coordY(m) \land coordZ(n) = coordZ(m)$

**Fig. 10.** $M_1$: Static Part 3

In order to model the transferring of messages through the communication channels, the variables *channel_state* and *channel_content* are defined in the second model to represent the control and the data value on each channel. Each channel can have the *busy* or $free$ state. When the channel receives data, its state switches from $free$ to *busy* and the message is added to the *channel_content*. When the channel transfers data to the end, *channel_state* changes to $free$ and the channel is released by removing the message from *channel_content*. The invariants of $M_1$ model that, when a channel is released, then its content is empty and can thus receive the next message; when a channel is busy, the message is in the channel. We illustrate these invariants in Fig.11.

**VARIABLES**
channel_content channel_state
**INVARIANTS**
$channel_content \in CHANNELS \rightarrowtail\!\!\!\!\!\!+ MESSAGES$
$dom(channel_content) = channel_state^{-1}[\{busy\}]$
$channel_state \in CHANNELS \rightarrow state$
$dom(channel_state) = ran(mk_channel)$
$ran(channel_content) \subseteq dom(moving_pool)$
$\forall msg \cdot msg \in dom(moving_pool) \wedge des(msg) = moving_pool(msg)$
$\Rightarrow msg \notin ran(channel_content)$

**Fig. 11.** $M_1$: Invariants (channels)

The *switching* event is now refined to transfer a message to the next router through channels. In order to model this, we add a new event *out_to_channel* as shown in Fig.15 (Appendix) to model pushing a message in the channel. This event is enabled when there is a message for transferring in a node and the channel between the node and the next node is free. In addition, we refine the *switching* event as shown in Fig.16 (Appendix) to model releasing the channel by receiving the message at the end of the channel. This event is enabled when a message is in the channel.

### 3.3 The Third Model $M_2$

In this model, we define buffers for the ports of the nodes and refine the second model to model the communication in 3D NoCs by considering these buffers.

**The Static Part.** The context of the third model contains a single constant $buffer_size \in \mathbb{N}_1$, which is a strict natural number denoting the maximum number of messages allowed in a buffer.

**The Dynamic Part.** Each node has fourteen buffers, each assigned to node ports; those assigned to output ports are called *output buffers* and those assigned to input ports are called *input buffers*. When there is a message in an output buffer of a node, the node can transfer it to the channel provided that the channel is free. If in the other side of the channel the input buffer has an empty place, the message is transferred to the input buffer of the next node and the channel is released; otherwise, the channel will be busy until an empty place appears in the input buffer. To model the buffer structure in the third model we add a new machine variable $buffer_content$ that models the current content of all buffers. Indeed, adding and removing messages in/from buffers is modeled by the $buffer_content$ variable.

In order to guarantee the correctness of the buffer modeling, we need the invariants shown Fig.12. They model that the content of a buffer never becomes more than its size. In addition, while a message is in the *moving_pool*, i.e., it has not reached to its destination, it must be either in a channel or in a buffer.

**VARIABLES**
buffer_content
**INVARIANTS**
$buffer_content \in MESSAGES \rightarrow\!\!\!\!\!\!+ (ROUTER \times PORTS)$
$dom(buffer_content) \cup ran(channel_content) = dom(moving_pool)$
$dom(buffer_content) \cap ran(channel_content) = \varnothing$
$\forall b \cdot b \in ran(buffer_content) \Rightarrow card(buffer_content \rhd \{b\}) \in 1 .. buffer_size$

**Fig. 12.** $M_2$: Invariants (buffer)

The *switching* event, as shown in Fig.17 (Appendix), is refined to be enabled when there is an input buffer with at least one empty place at the end of the channel. Then, besides releasing the channel, the message in the channel is transfered to the input buffer. The status of the *switching* event is still anticipated since we do not store. The *out_to_channel* event, as shown in Fig.17, is refined to be enabled when there is a message in an output buffer meaning that the message is removed from buffer. The *sent_message* and *received_message* events are refined so that they update the $buffer_content$ variable as shown in Fig. 18 (Appendix).

**Event** *routing* $\widehat{=}$
**extends** *routing*
**any**
*msg* *router* *in_p* *out_p*
**where**
$in_p \in \{Win, Ein, Sin, Nin, Uin, Din, Lin\}$
$out_p \in \{Wout, Eout, Sout, Nout, Uout, Dout, Lout\}$
$(in_p = Win \wedge out_p \neq Wout) \vee (in_p = Ein \wedge out_p \neq Eout)$
$\vee (in_p = Sin \wedge out_p \neq Sout) \vee (in_p = Nin \wedge out_p \neq Nout)$
$\vee (in_p = Uin \wedge out_p \neq Uout) \vee (in_p = Din \wedge out_p \neq Dout)$
$\vee (in_p = Lin \wedge out_p \neq Lout)$
$msg \mapsto (router \mapsto in_p) \in buffer_content$
$card(buffer_content \rhd \{router \mapsto out_p\}) < buffer_size$
**then**
$buffer_content(msg) := router \mapsto out_p$
**end**

**Fig. 13.** $M_2$: Routing Event

At this level of abstraction, we refine the *routing* event (modeled as *skip* in the previous models) as shown in Fig.13. A routing algorithm decides on choosing an output channel for a message in an input channel. As we present a general model, we do not consider any specific routing algorithm and we model routing decision nondeterministically. That is, when there is a message in an input buffer of a node, it can be routed to any output buffer of the node except the output buffer in the same direction with the input buffer e.g., a message in the northern input buffer cannot be routed to the northern output buffer. We also check that there is enough space in the chosen buffer. We have this constraint to prevent a cycling problem in the communication that would lead to deadlock in the interconnection network.

We do not change the status of the *switching* event in this refinement step. Thus, its status is still anticipated. In order to have it convergent, we need to define a variant based on some ordering relation of the message identifiers. This can be achieved by modeling a channel dependency graph but is not part of this paper.

A more concrete 3D NoC design can be modeled by refining one or more of these three general models and by verifying whether the design can guarantee the overall communication correctness. In the following, we model the XYZ routing algorithm by refining the third model $M_2$ and verifying whether it guarantees the overall communication correctness.

## 4 Case Study: The XYZ Routing Algorithm

In this section, we formally develop a dimension-order routing (DOR) algorithm which is a deterministic routing scheme widely used for NoCs [24]. To make the best use of the regularity of the topology, the dimension-order routing transfers packets along minimal paths in the traversing of the low dimension first until no further move is needed in this dimension. Then, they go along the next dimension and so forth until they reach their destination. For example, the dimension-order routing in the 3D NoC called the *XYZ routing algorithm* uses Z dimension channels after using Y and X dimension channels. Packets travel along the X dimension, then along the Y dimension and finally along the Z dimension. Thus, if $current_node = (c_x, c_y, c_z)$ is a node containing a message addressed to node $destination = (d_x, d_y, d_z)$, then the XYZ routing function $R_{xyz}(,)$ is defined as follows:

$$R_{xyz}((c_x, c_y, c_z), (d_x, d_y, d_z)) = \begin{cases} (c_{x-1}, c_y, c_z) & \text{iff} \quad c_x > d_x \\ (c_{x+1}, c_y, c_z) & \text{iff} \quad c_x < d_x \\ (c_x, c_{y-1}, c_z) & \text{iff} \quad c_x = d_x \wedge c_y > d_y \\ (c_x, c_{y+1}, c_z) & \text{iff} \quad c_x = d_x \wedge c_y < d_y \\ (c_x, c_y, c_{z-1}) & \text{iff} \quad c_x = d_x \wedge c_y = d_y \wedge c_z > d_z \\ (c_x, c_y, c_{z+1}) & \text{iff} \quad c_x = d_x \wedge c_y = d_y \wedge c_z < d_z \end{cases}$$

In order to model the XYZ routing algorithm based on the third general model, we have to refine the routing event which is nondeterministically defined. As shown in the above formula, a message can be transfered to six different directions based on its current position and destination. Therefore, we refine the *routing* event in the previous model to six *routing* events so that their guards are based on the routing formula. As an example of the *routing* event, we show in Fig.14 the situation where $c_x$ is greater than $d_x$. All the correctness properties defined for the abstract models are proved. Hence, the XYZ routing algorithm guarantees the overall communication correctness.

```
Event   routing_X_dec ≙
extends  routing
     where
          coordX(router) > coordX(des(msg)) ∧ out_p = Wout
     then
          buffer_content(msg) := router ↦ out_p
     end
```

**Fig. 14.** The XYZ Model: Routing Event ($c_x > d_x$)

## 5 Verification of the Models

In order to prove that the models satisfy their correctness properties we have to check that they respect their invariants, i.e., the pool properties for our models. To prove this, we have generated the proof obligations for all the models using the Rodin tool: part of the proof obligations were automatically discharged and the rest of could be proved interactively. The proof statistics for our models are shown in Table 1. These figures express the number of proof obligations generated by the Rodin platform as well as the number of obligations automatically discharged by the platform and those interactively proved. A high number of interactive proofs were due to reasoning about set comprehension and unions, not currently supported automatically in Rodin. In addition, the interactive proving often involved manually suggesting values to discharging various properties containing logical disjunctions or existential quantifiers. Extra proving was due to the fact that currently, we cannot create proof scripts and reuse them whenever needed in RODIN. Thus, in some cases we had to manually repeat very similar or almost identical proofs.

**Table 1.** Proof Statistics

| Model | Number of Proof Obligations | Automatically Discharged | Interactively Discharged |
|---|---|---|---|
| Context | 21 | 6(28%) | 15(72%) |
| $M_0$ Model | 38 | 34(89%) | 4(11%) |
| $M_1$ Model | 33 | 11(33%) | 22(67%) |
| $M_2$ Model | 33 | 7(21%) | 26(79%) |
| XYZ Model | 13 | 0(0%) | 13(100%) |
| Total | 144 | 64(45%) | 80(55%) |

## 6 Conclusions

In this paper, we have proposed the abstract models $M_0$, $M_1$, and $M_2$ at three increasing levels of detail for 3D NoCs. These can be used for modeling and verifying 3D NoC-designs in the early stages of the system development. We have also shown how to apply such an abstract model to verify a concrete 3D NoC routing algorithm. Most importantly, the overall correctness of the communication models (expressed using a special data structure called pool [2]) is guaranteed for the 3D NoCs. We have achieved this by modeling the correctness condition via invariants; as each model added detail to the previous model, the invariant needed to reflect these added details in a consistent manner. In order for the invariant to be satisfied by a model, a number of proof obligations needs to be discharged. Moreover, in order for the models to respect the refinement relation $M_0 \sqsubseteq M_1 \sqsubseteq M_2$, i.e., to develop each other in a provably

correct manner, some other proof obligations need to be generated. As we have employed the RODIN platform to specify our 3D NoC modeling, many of these proof obligations have been automatically discharged, while for the rest it was possible to discharge them interactively. We note an interesting property of our communication correctness condition, that essentially reduces to the fact that all the messages will eventually reach their destinations. This is a typical liveness property that we model here as an invariant, also based on the variant expression ensuring that our models will eventually terminate. The liveness property can also be verified via a model checker, for instance Pro-B [31], that is associated to the RODIN platform.

The NoC communication can be either unicast or multicast [23]. In the unicast communication a message is sent from a source node to a single destination node, while in the multicast communication a message is sent from a source node to an arbitrary set of destination nodes. We have considered here sending a message from a source to a single destination, hence modeled unicast communication. One of our future plans is to extend the abstract models $M_0$, $M_1$, and $M_2$ to also specify multicast communication and as a case study we target a novel routing protocol for multicast traffic called HAMUM [12], based on the extended 3D NoC model. HAMUM, Hamiltonian Adaptive path for both the Multicast and Unicast Model, is a new adaptive routing model based on Hamiltonian path for both the multicast and unicast traffic. An interesting property that we expect out of the multicast modeling is to have the case study reusing the $M_2$ model via an algorithmic refinement instead of a superposition one like we now have. This is because we can have several messages that could be routed in parallel using different events via several channels. Our XYZ routing employs already several events for the routing instead of the abstract routing event of the model $M_2$, but only one of them is enabled at all moments.

By strengthening the invariants we can verify more diverse properties of the 3D NoC designs, for instance we could prove deadlock-freedom for routing algorithms - currently, one of the most challenging properties for the 3D NoCs. For this, we envision an extension of the abstract 3D NoC models with an extra channel dependency graph to reason about deadlock-freedom; the HAMUM algorithm can then be shown as deadlock-free.

## References


1. Abrial, J.R.: A System Development Process with Event-B and the Rodin Platform. In: Butler, M., Hinchey, M.G., Larrondo-Petrie, M.M. (eds.) ICFEM 2007. LNCS, vol. 4789, pp. 1–3. Springer, Heidelberg (2007)
2. Abrial, J.R.: Modeling in Event-B: System and Software Design. Cambridge University Press (2010)
3. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press (1996)
4. Abrial, J.R., Cansell, D., Mery, D.: Refinement and Reachability in Even-B. In: 4th International Conference of B and Z Users, pp. 129–148 (2005)

5. Abrial, J.R., Hallerstede, S.: Refinement, Decomposition and Instantiation of Discrete Models: Application to Event-B. In: Fundamenta Informaticae, pp. 1–28 (2007)
6. Andreasson, D., Kumar, S.: Slack-Time Aware-Routing in NoC Systems. In: IEEE International Symposium on Circuits and Systems, pp. 2353–2356. IEEE (2005)
7. Arditi, L., Berry, G., Kishinevsky, M.: Late Design Changes (ECOs) for Sequentially Optimized Esterel Designs. In: Hu, A.J., Martin, A.K. (eds.) FMCAD 2004. LNCS, vol. 3312, pp. 128–143. Springer, Heidelberg (2004)
8. Back, R.J., Sere, K.: Stepwise Refinement of Action Systems. In: van de Snepscheut, J.L.A. (ed.) MPC 1989. LNCS, vol. 375, pp. 115–138. Springer, Heidelberg (1989)
9. Back, R.J., Sere, K.: Superposition Refinement of Reactive Systems. Formal Aspects of Computing 8(3), 324–346 (1996)
10. Borrione, D., Helmy, A., Pierre, L., Schmaltz, J.: A Formal Approach to the Verification of Networks on Chip. EURASIP Journal on Embedded Systems 2009(1), 1–14 (2009)
11. Duan, X., Zhang, D., Sun, X.: A Condition of Deadlock-free Routing in Mesh Network. In: Second International Conference on Intelligent Networks and Intelligent Systems, pp. 242–245 (2009)
12. Ebrahimi, M., Daneshtalab, M., Liljeberg, P., Tenhunen, H.: HAMUM A Novel Routing Protocol for Unicast and Multicast Traffic in MPSoCs. In: The 18th Euromicro Conference on Parallel, Distributed and Network-Based Computing (2010)
13. Feero, B.S., Pande, P.: Networks-on-Chip in a Three-Dimensional Environment: A Performance Evaluation. IEEE Transactions on Computers, 32–45 (2009)
14. Grecu, C., et al.: A Scalable Communication-Centric SoC Interconnect Architecture. In: 5th International Symposiom Quality Electronic Design (ISQED 2004), pp. 343–348 (2004)
15. Gupta, R., Guernic, P.L., Skuhla, S.K.: Formal methods and models for system design: a system level perspective. Kluwer Academic Publishers (2004)
16. Harrison, J.: Formal Verification at Intel. In: Symposium on Logic in Computer Science (2003)
17. Jerger, N.E., Peh, L.S., Lipasti, M.H.: Virtual Circuit Tree Multicasting: A Case for On-Chip Hardware Multicast Support. In: International Conference Computer Architecture, China, pp. 229–240 (2008)
18. Katz, S.: A Superimposition Control Construct for Distributed Systems. ACM Transactions on Programming Languages and Systems, 337–356 (1993)
19. Kaivola, R., Ghughal, R., Narasimhan, N., Telfer, A., Whittemore, J., Pandav, S., Slobodová, A., Taylor, C., Frolov, V., Reeber, E., Naik, A.: Replacing Testing with Formal Verification in Intel® CoreTM i7 Processor Execution Engine Validation. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 414–429. Springer, Heidelberg (2009)
20. Kim, Y.B., Kim, Y.-B.: Fault-Tolerant Source Routing for Networks-on-Chip. In: 22nd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 12–20. IEEE Computer Society (2007)
21. Liao, W., Hsiung, P.: Creating a Formal Verification Platform for IBM CoreConnect-based SoC. In: The 1st International Workshop on Automasted Technology for Verificatin and Analysis (ATVA 2003), pp. 7–18 (2003)
22. Loi, I., Benini, L.: An Efficient Distributed Memory Interface for Many-Core Platform with 3D Stacked DRAM. In: Proc. of the DATE Conference, Germany, pp. 99–104 (2010)

23. Lu, Z., Yin, B., Jantsch, A.: Connection-Oriented Multicasting in Wormhole-Switched Networks on Chip. In: Emerging VLSI Technologies and Architectures, pp. 205–211 (2006)
24. Montaana, J.M., Koibuchi, M., Matsutani, H., Amano, H.: Balanced Dimension-Order Routing for k-ary n-cubes. In: International Conference on Parallel Processing (2009)
25. Nayebi, A., Meraji, S., Shamaei, A., Sarbazi-azad, H.: XMulator: A listener-Based Integrated Simulation Platform for Interconnection Networks. In: Asia International Conference on Modeling and Simulation, pp. 128–132 (2007)
26. Palesi, M., Holsmark, R., Kumar, S., Catania, V.: Application Specific Routing Algorithms for Networks on Chip. IEEE Transactions on Parallel and Distributed Systems, 316–330 (2009)
27. Park, D., et al.: Mira, A Multi-Layered On-Chip Interconnect Router Architecture. In: ISCA 2008, pp. 251–261 (2008)
28. Tsiopoulos, L., Walden, M.: Formal Development of NoC Systems in B. Nordic Journal of Computing, 127–145 (2006)
29. Yan, S., Lin, B.: Design of Application-Specific 3D Networks-on-Chip Architectures. In: IEEE International Conference on Computer Design (ICCD 2008), pp. 142–149 (2008)
30. RODIN Tool Platform, http://www.event-b.org/platform.html
31. ProB Model Checker, http://www.stups.uni-duesseldorf.de/ProB/overview.php

# Appendix

```
Event   out_to_channel_ ≙
any new_position   current_msg   out_p   in_p
where
     current_msg ∈ dom(moving_pool) ∧ new_position ∈ POSITION
     moving_pool(current_msg) ↦ new_position ∈ Neigh
     out_p ∈ {Nout, Sout, Wout, Eout, Uout, Dout} ∧ in_p ∈ {Nin, Sin, Win, Ein, Uin, Din}
     (moving_pool(current_msg) ↦ out_p) ↦ (new_position ↦ in_p) ∈ dom(mk_channel)
     channel_state(mk_channel((moving_pool(current_msg)↦out_p)↦(new_position↦in_p))) = free
     moving_pool(current_msg) ≠ des(current_msg) ∧ current_msg ∉ ran(channel_content)
then
     channel_state(mk_channel((moving_pool(current_msg)↦out_p)↦(new_position↦in_p))) := busy
     channel_content(mk_channel((moving_pool(current_msg) ↦ out_p) ↦ (new_position ↦ in_p)))
                := current_msg
end
```

**Fig. 15.** $M_1$: Out_to_Channel Event

```
Event   switching ≙
Status  anticipated
extends  switching
any current_msg   new_position   p1   p2
where
     p1 ∈ {Nout, Sout, Wout, Eout, Uout, Dout} ∧ p2 ∈ {Nin, Sin, Win, Ein, Uin, Din}
     (moving_pool(current_msg) ↦ p1) ↦ (new_position ↦ p2) ∈ dom(mk_channel)
     channel_state(mk_channel((moving_pool(current_msg)↦p1)↦(new_position↦p2))) = busy
     current_msg =
       channel_content(mk_channel((moving_pool(current_msg) ↦ p1) ↦ (new_position ↦ p2)))
then
     moving_pool(current_msg) := new_position
     channel_state(mk_channel((moving_pool(current_msg)↦p1)↦(new_position↦p2))) := free
     channel_content := channel_content ⩥ {current_msg}
end
```

**Fig. 16.** $M_1$: Switching Event

```
Event   switching ≙
Status  anticipated
extends  switching
     where
          card(buffer_content ▷ {new_position ↦ p2}) < buffer_size
     then
          buffer_content := buffer_content ∪ {current_msg ↦ (new_position ↦ p2)}
     end
Event   out_to_channel ≙
extends  out_to_channel_
     where
          card(buffer_content ▷ {new_position ↦ in_p}) > 0
     then
          buffer_content := buffer_content \ {current_msg ↦ (moving_pool(current_msg) ↦ out_p)}
     end
```

**Fig. 17.** $M_2$: Switching and Out_to_Channel Events

```
Event   sent_message ≙
extends  sent_message
     where
          current_msg ∉ dom(buffer_content)
     then
          buffer_content := buffer_content ∪ {current_msg ↦ (src(current_msg) ↦ Lin)}
     end
Event   received_message ≙
extends  received_message
     where
          current_msg ↦ (des(current_msg) ↦ Lout) ∈ buffer_content
     then
          buffer_content := buffer_content \ {current_msg ↦ (des(current_msg) ↦ Lout)}
     end
```

**Fig. 18.** $M_2$: Sent_message and Received_message Events

# Paper VI

## Formal Modeling of Multicast Communication in 3D NoCs

**Maryam Kamali, Luigia Petre, Kaisa Sere and Masoud Daneshtalab**

Originally published in: P. Kitsos and S. Niar (Eds.), *Proceedings of the 14th Euromicro Conference on Digital System Design - DSD 2011*, pp. 634-642. IEEE/Euromicro, August 2011.

2011 14th Euromicro Conference on Digital System Design

# Formal Modeling of Multicast Communication in 3D NoCs

Maryam Kamali*†, Luigia Petre*, Kaisa Sere* and Masoud Daneshtalab‡
*Åbo Akademi University, Finland
Email: {maryam.kamali,luigia.petre,kaisa.sere}@abo.fi
†Turku Centre for Computer Science (TUCS), Finland
‡University of Turku, Finland
Email: masdan@utu.fi

***Abstract*—A reliable approach to designing systems is by applying formal methods, based on logics and set theory. In formal methods refinement based, we develop the system models stepwise, from an abstract level to a concrete one by gradually adding details. Each detail-adding level is proved to still validate the properties of the more abstract level. Due to the high complexity and the high reliability requirements of 3D NoCs, formal methods provide promising solutions for modeling and verifying their communication schemes. In this paper, we present a general model for specifying the 3D NoC multicast communication scheme. We then refine our model to two communication schemes,: unicast and multicast, via the XYZ routing algorithm in order to put forward the correct-by-construction concrete models.**

***Index Terms*—Formal methods; Event-B; refinement; 3D NoC; multicast communication**

## I. Introduction

The Network-on-Chip (NoC) architecture paradigm, based on a modular packet-switching mechanism, can address many of the on-chip communication design issues such as performance limitations of long interconnects and the integration of high numbers of Intellectual Property (IP) cores on a chip. However, the 2D-chip fabrication technology faces many challenges in the deep submicron regime even when employing NoC architectures. Examples of these problems are the design of the clock-tree network for large chips, limited floor-planning choices, the increase of both the wire delay and power consumption, the integration of various components that are digital, analog, MEMS and RF, etc. Three Dimensional Integrated Circuits (3D ICs) have been emerging as an attractive candidate, for the reduction of interconnection length and the added interconnect scalability in the third dimension offer an opportunity to further improve the performance of NoC [13], [15], [14].

Three dimensional Networks-on-Chips (3D NoCs) [13] provide more reliable interconnections due to the increased number of links between components. Due to their promise of parallelism and efficiency, 3D NoCs have a critical role in leading towards reliable computing platforms. However, the majority of their evaluation approaches are simulation-based tools, such as XMulator [16], Noxim [17], etc. Simulation-based approaches are usually applied in the late stages of design and are limited, e.g., by the length of time that a system is simulated. This means that exhaustive checking of all the system states is impossible in practice for complex 3D NoCs and thus, simulation is not suitable for verifying the correctness of a NoC design.

Another approach to address this problem is via formal methods. Formal methods refer to the application of mathematical techniques to the design and implementation of computer hardware and software. Prominent examples of applying formal methods are provided by, e.g., Intel [19], [18] and IBM [21] for formally verifying hardware or systems-on-chip (SoC) [20]. By using rigorous mathematical techniques, it is possible to deliver *provably correct* systems. Formal methods are based on the capture of system requirements in a specific, precise format. Importantly, such a format can be analyzed for various properties and, if the formal method permits, also stepwise developed until an implementation is formed. By following such a formal development, we are *sure* that the final result correctly implements the requirements of the system.

A general formal model of 3D NoCs was recently introduced in [12], to verify the design of unicast communication approaches. Unicast communication refers to sending a message from a single source node to a single destination node. In this paper we extend the proposed model to a general formal model of 3D NoCs for specifying and verifying the design of multicast communication. We show that the unicast model in [12] is an instance of our general model in this paper.

Communication in network-based multicore architectures can be either unicast (one-to-one) or multicast (one-to-many) [22]. While in the former approach a message is sent from a source node to a single destination node, in the latter approach a message is sent from a source node to an arbitrary set of destinations. Multicast communication is employed in many multicore applications, e.g., replication, barrier synchronization, cache coherency in distributed shared-memory architectures, and clock synchronization [22], [23]. Multicast routing algorithms can be classified as unicast-based, tree-based, and path-based [25]. The first alternative produces much unnecessary traffic, also increasing the latency and congestion of the network [25]. The path-based method forces packets to follow longer routes than what is proposed in the tree-based method, hence the tree-based method seems the most efficient of the three [26].

978-0-7695-4494-6/11 $26.00 © 2011 IEEE
DOI 10.1109/DSD.2011.86

IEEE computer society

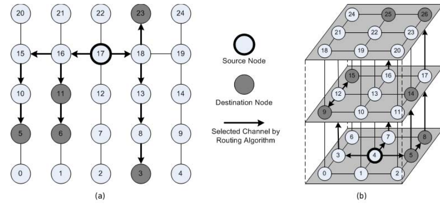



Fig. 1. Example of tree-based multicast routing in (a) 2D-mesh and (b) 3D-mesh

The contribution of our paper consists in proposing a general formal model for 3D NoC multicast communication. This model can be employed to develop various provably correct communication schemes in 3D NoCs. More importantly, this general model can be used for analysis purposes, as a common basis for comparing different communication schemes. For instance, we could employ our modeling to prove that the tree-based multicast algorithm is more efficient than the path-based algorithm.

We proceed as follows. In Section II we briefly present the tree-based multicast routing idea as well as describe our formal method of choice, namely Event-B, to the extend needed in this paper. In Section III we describe our three, stepwise-constructed formal models for multicast communication. We apply the most concrete model to two communication schemes in Section IV. In Section V we shortly discuss the proving methodology and in Section VI we conclude.

## II. Preliminaries

### A. Tree-based Multicast Communication

In tree-based routing, the destination set is partitioned at the source and separate copies of the message are sent through one or more outgoing channels. A message may be replicated at intermediate nodes and forwarded toward disjoint subsets of destinations in the tree through multiple output channels. As shown in Fig. 1(a), the source node 17 is selected as the root and a spanning tree is formed with respect to the root; when messages enter routers at branch points (nodes 16 and 18), they are duplicated and forwarded to multiple output channels. Similarly, the branch nodes of the 3D-mesh in Fig. 1(b) are nodes 5 and 12.

### B. Event-B

Event-B [2], [1] is a formal framework derived from the B Method [4], [3] to model and reason about parallel, distributed and reactive systems. Event-B has the associated Rodin platform [6] which provides automated tool support for modelling and verification by theorem proving. The Rodin platform [1], [6] generates a set of proof obligations in order to show the correctness of formal models. Then it automatically discharges part of the proof obligations and also provides the means for the user to discharge interactively the remaining proofs. In Event-B, a system model is gradually specified on increasing levels of abstraction, always ensuring that a more concrete model is a *correct implementation* of an abstract model.

In Event-B, a system specification (model) is defined using the notion of an *abstract state machine* [5]. An abstract state machine encapsulates the model state, represented as a collection of model variables, and defines operations on this state, called events. Thus, it describes the behaviour of the modelled system, also referred to as the dynamic part. The system properties that should be preserved during the execution are formulated as a list of *invariant* predicates over the state of the model. A machine may also have an accompanying component, called *context*, which contains the static part of the system. A context can include user-defined carrier sets, constants and their properties, given as a list of model axioms.

An event, modeling state changes, is composed of a *guard* and an *action*. The guard is the necessary condition under which an event might occur, i.e., when the event is *enabled*. If several events are enabled at the same time, any of them can be chosen for execution nondeterministically. The action determines the way in which the state variables change when the event occurs. For initializing the system, a sequence of actions is defined. If some events have no variables in common and are enabled at the same time, then they can be considered to be executed in parallel since their sequential execution in any order gives the same result. For all practical purposes, this execution model is parallel and can be implemented as such when the model is refined to code. Events can be declared as *anticipated*, meaning that in the future refinements we need to set out a natural number expression called *variant* and prove that it is decreased by this event. Events can also be *convergent*, meaning that in the current machine there is a variant that decreases when this event is chosen for execution. Thus, an anticipated event is not convergent in the current machine but should become so in a future refinement of that machine.

A model can be developed by a number of correctness preserving steps called *refinements* [7], [8]. One form of model refinement can add new data and new events on top of the already existing data and behavior but in such a way that the introduced behavior does not contradict or take over the abstract machine behavior. This form of stepwise construction is referred to as *superposition* refinement [10], [11]. We may also use other refinement forms, e.g., *algorithmic* refinement [9]. In this case, an event of an abstract machine can be refined by several corresponding events in a refined machine. This will model different branches of execution, that can, for instance, take place in parallel and thus can improve the algorithmic efficiency.

The 3D NoC designs are quite complex in the state size, given by the number of variables, as well as in the number of transitions. Thus, it is very complicated to describe the whole system as one model. One feasible methodology to address this modeling problem is given by the refinement techniques, by modeling at different abstraction levels. This methodology is valuable for at least two reasons. First, the idea of stepwise construction provides the gradual development of the designs, ensuring their correctness at every development step. Second,

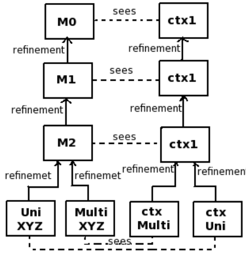



Fig. 2. Model development for multicast communication

the abstract models can contain non-determinism, which allows that in the subsequent models we simply fix the choice to one specific execution. Obviously, in this case, we can develop alternative models corresponding to different (fixed) choices, all refining the same abstract model.

## III. Three Abstract Models for multicast communication

In this section we formally develop three high-level models $M_0$, $M_1$, and $M_2$ for the 3D NoC. Our models are at three increasing levels of detail so that each model is a refinement of the previous one: $M_0 \sqsubseteq M_1 \sqsubseteq M_2$ as shown in Fig. 2. We start by building a model that is far more abstract than the final network we want to construct. The idea is to take into account initially only very few constraints. This is because we want to be able to reason about this system in a simple way, considering in turn each element of the network. Therefore, in the initial model, we specify a network of nodes as a context and define the correctness properties of this network based on a specific data structure called *pool*, as suggested by [2]. In the second model, we refine the initial model by adding new data and events to model the 3D mesh-based NoC architecture; besides, we specify the channels between nodes. In the third model, we model buffers for nodes and refine the communication model by considering the correctness properties of the network. This design strategy lets us add complexity to the model based on the correctness properties of the network, defined abstractly already in the initial model.

### A. The initial model $M_0$

The first model $M_0$ that we construct is rather abstract: we do not consider the numerous parts of the network such as channels or buffers; they will be introduced in subsequent refinements. $M_0$ will thus allow us to reason about the system very abstractly [2]. The model $M_0$ is formed of the static part (ctx1) and the dynamic part (M0), as follows.

The static part of our model describes the network structure in terms of neighbors and the messages to travel through the network. We model the network structure with a relation *Neigh* on the finite and non-empty *ROUTER* set; this relation denotes the communication links among nodes, expressed with the many to many mapping relation operator $\leftrightarrow$. We model the messages to travel through the network with a finite, non-empty set *MESSAGES*. A message is modeled as a triple *(data, source, destinations)*, where *data* is an element of the finite and non-empty *DATA* set (modeling the information to transmit), *source* is an element of the set *ROUTER* where a message is injected, and *destinations* is a subset of the set *ROUTER* where a message should be received. We define two functions, named $src$ and $des$ that return source and destinations, respectively. The detailed properties of these structures are formally introduced in [12], where they only had a different type of destination. The modification we need for modeling multicast refers to defining the $MESSAGES$ set, expressed with the total function operator $\rightarrow$, as follows:

$$MESSAGES \in DATA \rightarrow ROUTER \rightarrow \mathbb{P}(ROUTER)$$
$$Neigh \in ROUTER \leftrightarrow ROUTER$$
$$src \in MESSAGES \rightarrow ROUTER$$
$$des \in MESSAGES \rightarrow \mathbb{P}(ROUTER)$$

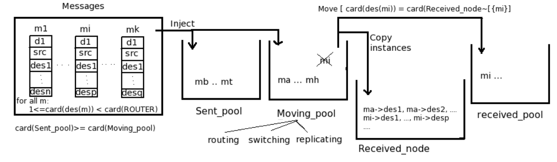



Fig. 3. Multicast Message Pools

In our network model we use the following condition for modeling the communication correctness: the messages in the network will eventually reach their destinations. To implement this condition in our multicast communication scheme, we define two message subsets, one message-to-node relation and one node-to-message relation as machine variables: $sent_pool \subseteq MESSAGES$, $received_pool \subseteq MESSAGES$, $moving_pool \in sent_pool \leftrightarrow ROUTER$ and $received_node \in ROUTER \leftrightarrow sent_pool$.

The map $moving_pool$ models the current position of messages which are routed in the network. In [12], there is only one instance of a message in the network; therefore, $moving_pool$ is defined as a (partial) function. However, in this paper, we model $moving_pool$ as a relation of messages-to-nodes to show the existence of a number of instances of a message in different nodes. In order to show that a message is received to all its destinations, we define a new relation $received_node \in ROUTER \leftrightarrow MESSAGES$ which models instances of messages having reached destinations. When a message is received to one of its destinations, a corresponding map is added to the $received_node$ relation. In order to have a message in $received_pool$, which denotes the list of received messages, a destination-to-message map for all of the destinations of that message should be in $received_node$. The new message pool structure is illustrated in Fig. 3.

One of the working steps of multicast communication is to replicate a message in intermediate or destination nodes. When a message is received to a node the message can be consumed and forwarded to one or several immediate neighbors or consumed and removed from the network. Therefore, we can produce multiple instances of a message in a node and switch these instances to different neighbors. In order to

**Event** $sent_message$
  **any** $current_msg$
  **where**
    $current_msg \in MESSAGES$
    $current_msg \notin sent_pool$
  **then**
    $sent_pool := sent_pool \cup \{current_msg\}$
    $moving_pool := moving_pool \cup \{current_msg \mapsto src(current_msg)\}$
    $replicate_msg_num(current_msg \mapsto src(current_msg)) := 0$
    $replicate_flag(current_msg \mapsto src(current_msg)) := FALSE$
  **end**

Fig. 4. $M_0$: sent_message Event

**Event** $replicating$
  **any** $current_msg \quad pos \quad num$
  **where**
    $current_msg \mapsto pos \in moving_pool$
    $replicate_flag(current_msg \mapsto pos) = FALSE$
    $received_node^{-1}[\{current_msg\}] \cup \{pos\} \subset des(current_msg)$
    $num \in 0\,..\,6$
  **then**
    $replicate_flag(current_msg \mapsto pos) := TRUE$
    $replicate_msg_num(current_msg \mapsto pos) := num$
  **end**

Fig. 5. $M_0$: replicating Event

**Event** $switching$
  **any** $current_msg \quad pos \quad new_position$
  **where**
    $current_msg \mapsto pos \in moving_pool$
    $pos \notin des(current_msg)$
    $new_position \mapsto pos \in Neigh$
    $new_position \neq src(current_msg)$
    $replicate_flag(current_msg \mapsto pos) = TRUE$
    $replicate_msg_num(current_msg \mapsto pos) > 0$
    $current_msg \mapsto new_position \notin moving_pool$
    $current_msg \notin received_node[\{new_position\}]$
  **then**
    $moving_pool := moving_pool \cup \{current_msg \mapsto new_position\}$
    $replicate_flag := replicate_flag \cup$
      $\{current_msg \mapsto new_position \mapsto FALSE\}$
    $replicate_msg_num := (\{current_msg \mapsto pos\} \lhd replicate_msg_num)$
      $\cup \{current_msg \mapsto pos \mapsto (replicate_msg_num$
      $(current_msg \mapsto pos) - 1), current_msg \mapsto new_position \mapsto 0\}$
  **end**

Fig. 6. $M_0$: switching Event

model the message replication in nodes, we define two new variables: $replicate_flag \in moving_pool \rightarrow BOOL$ and $replicate_msg_num \in moving_pool \rightarrow 0..n-1$ (where $n$ is the maximum number of node neighbors; we assume this is a constant of the model). When $replicate_flag$ returns *false* for a pair of (node, message), this means that the message is received to a node but it is not replicated yet. When $replicate_flag$ returns *true* for a pair of (node, message), this models that the message which is received to the node is replicated. In this case, the number of message replicas in the node is expressed with $replicate_msg_num$.

To model the communication and the message pool functions, we add five more events to the unicast model than previously developed in [12]. In the following, we summarize the already defined events with the changes as well as the new events. The $sent_message$ event handles the injection of a new message into the network. As shown in Fig. 4, whenever a message is injected into the network, both $sent_pool$ as well as $moving_pool$ are updated. Moreover, the pair of (source node, message) with the value *false* is added to the $replicate_flag$ to show the existence of a message in the source node with no replication yet.

A message in $moving_pool$ should be routed toward its destination. This is composed of three actions, one for deciding which node would be the next one (routing), the other for replicating a message to a number of instances of the message (replicating), and the last for transferring the message to that node (switching). These three actions are available for all the nodes, including the source, the destination as well as all the intermediate nodes and are modeled respectively by the $routing$, $replicating$, and four $switching$ events. The $routing$ event consists of $skip$ at this abstract level as routing decisions are modeled only when a particular routing algorithm is needed to be verified; we then add the routing as a refinement to this abstract model.

The $replicating$ event shown in Fig. 5 handles the replication of a message in a node. When a message arrives to a node, the $replicating$ event becomes enabled for that node and the message and a random number between 0 and n-1 is assigned to the pair (node, message) in $replicate_msg_num$. The assigned number is nondeterministically chosen, because it makes our initial model general and reusable for a variety of communication design schemes. The number will be assigned deterministically when the model is refined to a particular communication design. We illustrate this in Section IV with the XYZ routing case study. When the number of replication for a message in a node is selected, the instances of the message should be transfered from the current node to its neighbors nondeterministically and the $moving_pool$, $received_node$, $replicate_msg_num$ and $replicate_flag$ for the message instances should be updated.

Because different states are possible for switching, the switching function is splitted in four switching events: $Switching$, $switching_removing$, $switching_garbage$ and $switching_removing_garbage$.

Two possible states of switching occur when the transferring message is not the last replicated instance and when the transferring message is the last one. When it is not the last instance, the $switching$ event as illustrated in Fig. 6 transfers the message to the next node by adding the pair (next node, message) to the $moving_pool$ and decreasing the number of replicated instances in the current node. The $replicate_flag$ of the message and the new position are updated to *false*. The *false* value of $replicate_flag$ for a message and a node expresses the receipt of a new message. When a message is the last replicated instance of a message in a node, the $switching_removing$ event illustrated in Fig. 7 transfers the message to the next node by adding (next node, message) and removing the previous (node, message) pairs to/from $moving_pool$. In addition, the previous (node, message) pair is removed from $replicate_flag$ and $replicate_msg_num$, expressed with the domain subtraction operator $\lhd$.

If the next node has already received a message from its other neighbors but the same message is switch-

**Event** $Switching_removing$
**any** $current_msg \quad new_position \quad pos$
**where**
$current_msg \mapsto pos \in moving_pool$
$pos \notin des(current_msg)$
$new_position \mapsto pos \in Neigh$
$new_position \neq src(current_msg)$
$replicate_flag(current_msg \mapsto pos) = TRUE$
$replicate_msg_num(current_msg \mapsto pos) = 0$
$current_msg \mapsto new_position \notin moving_pool$
$current_msg \notin received_node[\{new_position\}]$
**then**
$moving_pool := (moving_pool \setminus \{current_msg \mapsto pos\}) \cup \{current_msg \mapsto new_position\}$
$replicate_flag := (\{current_msg \mapsto pos\} \lhd replicate_flag) \cup \{current_msg \mapsto new_position \mapsto FALSE\}$
$replicate_msg_num := (\{current_msg \mapsto pos\} \lhd replicate_msg_num) \cup \{current_msg \mapsto new_position \mapsto 0\}$
**end**

Fig. 7. $M_0$: switching_removing Event

**Event** $switching_garbage$
**any** $current_msg \quad pos \quad new_position$
**where**
$current_msg \mapsto pos \in moving_pool$
$pos \notin des(current_msg)$
$new_position \mapsto pos \in Neigh$
$new_position \neq src(current_msg)$
$replicate_flag(current_msg \mapsto pos) = TRUE$
$replicate_msg_num(current_msg \mapsto pos) > 0$
$current_msg \mapsto new_position \in moving_pool \lor current_msg \in received_node[\{new_position\}]$
**then**
$replicate_msg_num := (\{current_msg \mapsto pos\} \lhd replicate_msg_num) \cup \{current_msg \mapsto pos \mapsto (replicate_msg_num(current_msg \mapsto pos) - 1), current_msg \mapsto new_position \mapsto 0\}$
**end**

Fig. 8. $M_0$: switching_garbage Event

ing in the next node, then the message from the previous node is removed from the network since it represent duplication. The $switching_garbage$ (Fig.8) and $switching_removing_garbage$ (Fig. 9) events handle the same situations as $switching$ and $switching_removing$ events with the difference that the received message to a next node is not added to the $moving_pool$ because the message is an extra instance of a message.

The $received_message$ event in the unicast model adds a message received at its destination to $received_pool$ and removes the message from $moving_pool$. This event is splitted into two receiving events in the multicast model: one event models the case when a message is received to all its destinations and one events models the case when a message is received to one of its destinations and it still has destinations to reach. The $received_message_complete$ event handles the former case and the $received_message$ event handles the latter case.

**Event** $switching_removing_garbage$
**any** $current_msg \quad pos \quad new_position$
**where**
$current_msg \mapsto pos \in moving_pool$
$pos \notin des(current_msg)$
$new_position \mapsto pos \in Neigh$
$new_position \neq src(current_msg)$
$replicate_flag(current_msg \mapsto pos) = TRUE$
$replicate_msg_num(current_msg \mapsto pos) = 0$
$current_msg \mapsto new_position \in moving_pool \lor current_msg \in received_node[\{new_position\}]$
**then**
$moving_pool := (moving_pool \setminus \{current_msg \mapsto pos\})$
$replicate_flag := (\{current_msg \mapsto pos\} \lhd replicate_flag)$
$replicate_msg_num := (\{current_msg \mapsto pos\} \lhd replicat_msg_num)$
**end**

Fig. 9. $M_0$: switching_removing_garbage Event

**Event** $received_message$
**any** $current_msg \quad pos$
**where**
$received_node^{-1}[\{current_msg\}] \cup \{pos\} \subset des(current_msg)$
$current_msg \mapsto pos \in moving_pool$
$replicate_flag(current_msg \mapsto pos) = TRUE$
$replicate_msg_num(current_msg \mapsto pos) > 0$
$pos \mapsto current_msg \notin received_node$
**then**
$received_node := received_node \cup \{pos \mapsto current_msg\}$
**end**

Fig. 10. $M_0$: received_message Event

**Event** $complete_received_message$
**any** $current_msg \quad pos$
**where**
$current_msg \mapsto pos \in moving_pool$
$pos \in des(current_msg)$
$received_node^{-1}[\{current_msg\}] \cup \{pos\} = des(current_msg)$
**then**
$received_node := received_node \cup \{pos \mapsto current_msg\}$
$moving_pool := \{current_msg\} \lhd moving_pool$
$received_pool := received_pool \cup \{current_msg\}$
$replicate_msg_num := (\{current_msg\} \lhd moving_pool) \lhd replicate_msg_num$
$replicate_flag := (\{current_msg\} \lhd moving_pool) \lhd replicate_flag$
**end**

Fig. 11. $M_0$: Complete_Received_message Event

The $received_message$ event illustrated in Fig. 10 adds a message received at one of its destinations to $received_node$ when at least one of the message destinations is not received yet. The $received_message_complete$ event, illustrated in Fig. 11 adds a message received at all its destinations to $received_pool$ and removes the message from $moving_pool$. A message is received to all its destinations when the same number of destinations is in $received_node$. This event is convergent: if new messages are not injected to the network for a certain time, all the messages will be received at their destinations. This is proved by means of the $(sent_pool \setminus received_pool)$ variant denoting the difference between the sets $sent_pool$ and $received_pool$.

In order to prove the communication correctness, we need to prove that $received_pool$ subset contains only messages which are received to all of their destinations. In addition, we have to prove that the $sent_pool$ subset eventually becomes equal with the $received_pool$ subset and the $moving_pool$ subset is empty when all the messages are received at their destinations. These properties are illustrated in Fig. 12 as invariants.

$M_0$ is a general specification of a general network and will be refined to model 3D NoC multicast communication designs in the following. Moreover, the model provides the necessary properties that should be preserved by refinement. These properties, that guarantee the overall communication

**Invariants**
$received_pool \cap dom(moving_pool) = \varnothing$
$received_pool \cup dom(moving_pool) = sent_pool$
$\forall msg \cdot msg \in dom(moving_pool) \Rightarrow$
$(\exists d \cdot d \in des(msg) \wedge d \mapsto msg \notin received_node)$
$\forall msg \cdot msg \in received_pool \Rightarrow$
$(\forall d \cdot d \in des(msg) \Rightarrow d \mapsto msg \in received_node)$
$\forall msg \cdot msg \in sent_pool \wedge des(msg) \setminus received_node^{-1}[\{msg\}] \neq \varnothing$
$\Rightarrow msg \in dom(moving_pool)$

Fig. 12. $M_0$: Invariants (Pool Modeling)

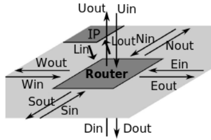



Fig. 13. Router in 3D Mesh-Based NoCs

correctness, are defined as the list of invariants.

*B. The Second Model $M_1$*

In the unicast model of communication presented in [12], the 3D NoC mesh-based architecture is specified by restricting the neighbor relation so that two nodes with coordinates $(x_i, y_i, z_i)$ and $(x_j, y_j, z_j)$ are neighbors if and only if $|x_i - x_j| + |y_i - y_j| + |z_i - z_j| = 1$. Based on this architecture and the seven-port routers, the communication channels between routers are defined by specifying a relation between output and input ports of neighbors as illustrated in Fig. 13.

To show how two neighbors are connected to each other through channels, the $def_channel \in (ROUTER \times PORTS) \rightarrow (ROUTER \times PORTS)$ relation is defined. The $def_channel$ relation models the relation of a port of a node to the corresponding port of its neighbor. As an illustration, east and west ports of neighbor nodes with different X coordinates are related to each other through a channel. As shown in Fig. 14, the $Ein$ and $Eout$ ports of node $(1,1,1)$ are connected to the $Wout$ and $Win$ ports of node $(2,1,1)$ through a channel, via relations $((1,1,1) \mapsto Eout) \mapsto ((2,1,1) \mapsto Win)$ and $((2,1,1) \mapsto Wout) \mapsto ((1,1,1) \mapsto Ein)$ in $def_channel$.

Based on these channel and neighbor structures given in the unicast model, we refine the initial model to specify the transfering of messages through the communication channels, so that the overall correctness of communication holds.

In order to show the data transfer through the channels, we specify two new variables in the second model: $channel_state \in CHANNELS \rightarrow state$ and $channel_content \in CHANNELS \rightarrowtail\!\!\!\!\!\!\rightarrow MESSAGES$. The $channel_state$ variable models the fact that a channel is either

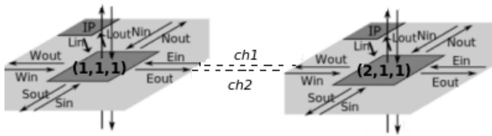



Fig. 14. Channels in 3D Mesh-Based NoCs

**Invariants**
$dom(channel_content) = channel_state^{-1}[\{busy\}]$
$ran(channel_content) \subseteq dom(moving_pool)$
$\forall msg \cdot msg \in dom(moving_pool) \wedge des(msg) = moving_pool(msg)$
$\Rightarrow msg \notin ran(channel_content)$

Fig. 15. $M_1$: Invariants (channels)

occupied by a message or free to transport a message; the variable $channel_content$ indicates which message is currently transferred via the channel. When a message is injected into a channel, its associated $channel_state$ variable is set to *busy* and the message is added to the $channel_content$ variable. We model the receipt of the message at the other end of channel by $channel_state$ being set to *free* and by removing the message from the $channel_content$ variable.

In order to construct a correct model in this refinement step, the model should satisfy the properties illustrated in Fig. 15 as well as the invariants defined the initial model. The new invariants model that, when a channel is released, its content is empty and can thus receive the next message; when a channel is busy, the message is in the channel.

In the initial model, transfering a message from a node to a neighbor node is modeled in one step while in this model it is refined to show it in two steps through channels: first, pushing a message in the channel is modeled by a newly introduced event $out_to_channel$ and second, releasing the channel by receiving the message at the end of the channel is modeled by extending the four switching events.

The guards of these events are strengthened with new conditions. Namely, these conditions check whether a channel starting from a node is *busy*, then the channel is released and the corresponding message is removed from the $channel_content$ and transfered to the other end of the channel, if the same message has not already been received there.

*C. The Third Model $M_2$*

In the third model, we define node buffers by attaching a buffer to each port of a router. A new variable $buffer_content \in MESSAGES \leftrightarrow (ROUTER \times PORTS)$ is defined to show the content of each buffer in a node. In the multicast model, several instances of a message can appear in several nodes, while in the unicast model only one instance of a message can appear in one node. Therefore, $buffer_content$ in unicast model is defined by a partial function while in this model we define it with a binary relation.

At this level of abstraction, we refine the $routing$ event (modeled as skip in the previous models) as shown in Fig. 16, to route a number of instances of a message in a node. A message from an input buffer is routed to a number of output buffers. The number of output buffers is equal to the number of instances for the pair (router, message) in the $replicate_msg_num$ relation. As we present a general model, we do not consider any specific routing algorithm and we model routing decisions nondeterministically. That is, when there is a message in an input buffer of a node, it can be routed to several output buffers of the node except the output buffer

**Event** $routing$
**any** $msg \quad router \quad in_p \quad out_p$
**where**
$in_p \in \{Win, Ein, Sin, Nin, Uin, Din, Lin\}$
$out_p \subseteq \{Wout, Eout, Sout, Nout, Uout, Dout, Lout\}$
$(in_p = Win \land Wout \notin out_p) \lor (in_p = Ein \land Eout \notin out_p) \lor$
$(in_p = Sin \land Sout \notin out_p) \lor (in_p = Nin \land Nout \notin out_p) \lor$
$(in_p = Uin \land Uout \notin out_p) \lor (in_p = Din \land Dout \notin out_p) \lor$
$(in_p = Lin \land Lout \notin out_p)$
$msg \mapsto (router \mapsto in_p) \in buffer_content$
$\forall port \cdot port \in out_p \Rightarrow card(buffer_content \rhd \{router \mapsto port\})$
$< buffer_size$
$replicate_flag(msg \mapsto router) = TRUE$
**then**
$buffer_content := buffer_content \cup (\{msg\} \times (\{router\} \times out_p))$
**end**

Fig. 16. $M_2$: Routing Event

$routing \in POSITION \times POSITION \rightarrow$
$\{Nout, Sout, Eout, Wout, Uout, Dout, Lout\}$
$\forall cur, dest \cdot cur \in POSITION \land dest \in POSITION \land$
$coordX(dest) > coordX(cur) \Rightarrow routing(cur \mapsto dest) = Eout$
$\forall cur, dest \cdot cur \in POSITION \land dest \in POSITION \land$
$coordX(dest) < coordX(cur) \Rightarrow routing(cur \mapsto dest) = Wout$
$\forall cur, dest \cdot cur \in POSITION \land dest \in POSITION \land$
$coordX(dest) = coordX(cur) \land coordY(dest) > coordY(cur)$
$\Rightarrow routing(cur \mapsto dest) = Nout$
$\forall cur, dest \cdot cur \in POSITION \land dest \in POSITION \land$
$coordX(dest) = coordX(cur) \land coordY(dest) < coordY(cur)$
$\Rightarrow routing(cur \mapsto dest) = Sout$
$\forall cur, dest \cdot cur \in POSITION \land dest \in POSITION \land$
$coordX(dest) = coordX(cur) \land coordY(dest) = coordY(cur)$
$\land coordZ(dest) > coordZ(cur) \Rightarrow routing(cur \mapsto dest) = Uout$
$\forall cur, dest \cdot cur \in POSITION \land dest \in POSITION \land$
$coordX(dest) = coordX(cur) \land coordY(dest) = coordY(cur) \land$
$coordZ(dest) < coordZ(cur) \Rightarrow routing(cur \mapsto dest) = Dout$
$\forall cur, dest \cdot cur \in POSITION \land dest \in POSITION \land$
$coordX(dest) = coordX(cur) \land coordY(dest) = coordY(cur) \land$
$coordZ(dest) = coordZ(cur) \Rightarrow routing(cur \mapsto dest) = Lout$

Fig. 17. The Unicast XYZ Routing Function

$new_des \in POSITION \times POSITION \times \mathbb{P}(POSITION) \rightarrow \mathbb{P}(POSITION)$
$\forall source, cur, dest \cdot source \in POSITION \land cur \in POSITION \land$
$dest \subseteq POSITION \land coordX(source) < coordX(cur) \Rightarrow$
$(\forall d \cdot d \in dest \land coordX(d) \geq coordX(cur) \Rightarrow$
$d \in new_des(source \mapsto cur \mapsto dest))$

Fig. 18. The $D_{+x,\pm y,\pm z}$ Subset of Destinations

in the same direction with the input buffer e.g., a message in the northern input buffer cannot be routed to the northern output buffer. We also check that there is enough space in the chosen buffer. We have this constraint to prevent a cycling problem in the communication that would lead to deadlock in the interconnection network.

The general functions of multicast communication in 3D NoC are specified in these three models. However, in order to present a general multicast communication model, a specific algorithm is not considered for replication. This allows us to model any multicast routing algorithms by refining these general models. To show that the unicast model is an instance of our model, in the following we refine the third model to model the unicast XYZ routing algorithm which is given as a case study in the unicast model [12]. Moreover, we also model the multicast XYZ routing algorithm by refining the third model, to show how these general models can be refined to a concrete multicast 3D NoC design.

## IV. Case Study: the Unicast and Multicast XYZ Routing Algorithms

As one of the contributions in this paper we show that our model is a generalization of the unicast model introduced in [12]. In the following, we formally develop the same deterministic routing algorithm as a case study, by refining our multicast model. Therefore, the unicast XYZ routing algorithm which is a dimension order routing in the 3D NoC is modeled. The XYZ routing algorithm uses Z dimension channels after using Y and X dimension channels. Packets travel along the X dimension, then along the Y dimension and finally along the Z dimension.

In order to model the unicast XYZ routing algorithm, we formally develop the routing function as a context part of our model and refine the $routing$ event in the third model so that the nondeterministic assignment to an output port becomes a deterministic assignment. This assignment is based on the XYZ routing function. A *routing* function as shown in Fig. 17 models the output port pair. This shows that unicast communication designs can be verified by our multicast model; moreover, we can also prove the correctness properties defined for the abstract models.

In order to show a case study in multicast communication designs, we extend the XYZ routing algorithm to a tree-based multicast algorithm in which the message is delivered to each destination in XYZ fashion. The multicast routing algorithm presented here is based on the concept of network partitioning. A multicast operation is implemented as several submulticasts, each destined for a subset of the destinations and each routed in a different subnetwork. In fact, the submulticasting recursively employs the concept of network partitioning so that, in each step of replicating and routing, the subnetwork and the subset of the destinations are partitioned again.

For a given set of destinations $D$, this is divided into at most six subsets in each node, $D_{+x,\pm y,\pm z}$, $D_{-x,\pm y,\pm z}$, $D_{x,-y,\pm z}$, $D_{x,+y,\pm z}$, $D_{x,y,+z}$, $D_{x,y,-z}$, according to the current position, the destinations and the source positions. The set $D_{+x,\pm y,\pm z}$ contains the destination nodes to the east of the current position when the current position is in the east of the source node; $D_{x,-y,\pm z}$ contains the destination nodes to the south of the current position when the current position is in the same $X$ coordinate as the source and in the south of source node; and so on. Formally, the subsets are described by defining a $new_des$ function in Event-B as a constant in the static part of our multicast XYZ routing algorithm model, as partially shown in Fig. 18. For a triple (source, current position, destinations), the output of the function is mapped to at most one of these subsets of destinations.

Based on the subsets of destinations ($D$), each node takes decisions about the number and direction of replicas of a message by constructing subnetworks. The new subsets of destinations partitions the network $N$ to subnetworks of $N_{x,y,+z}$, $N_{x,y,-z}$, $N_{+x,\pm y,\pm z}$, $N_{-x,\pm y,\pm z}$, $N_{x,-y,\pm z}$,

```
replicate ∈ POSITION × ℙ(POSITION)→
 ℙ({Nout, Sout, Eout, Wout, Uout, Dout})
∀cur, dest·cur ∈ POSITION ∧ dest ⊆ POSITION ∧ (∃d·d ∈ dest∧
 coordX(d) > coordX(cur)) ⇒ Eout ∈ replicate(cur ↦ dest)
```

Fig. 19. The east Replication Function

$N_{x,+y,\pm z}$. For instance, the subnetwork $N_{x,y,+z}$ contains the node $(i, j, k)$ which $i = current_x, j = current_y, k > current_z$ and the subset $D_{x,+y,\pm z}$ can be implemented in $N_{x,+y,\pm z}, N_{x,y,-z}, N_{x,y,+z}$; and so on. Thus in each branch node, the message is replicated to at most six subnetworks.

As shown in Fig. 19, the $replicate$ function implemented in each node gives the directions to replicate a message, for a pair of current position and the subset of destinations. The $replicate$ function checks the existence of a destination in subnetworks to select the directions on which the message should be routed. If the destination is in a subnetwork, the direction towards that subnetwork added to the list of replication. For instance, for the subset of destinations $D_{x,y,\pm z}$ that can be implemented in subnetwork $N_{x,y,-z}$ and/or $N_{x,y,+z}$, the result of the $replicate$ function can be North and/or South, respectively.

As an illustration, we explain the subnetworking and replication functions by the example of Fig. 1. If we consider the node 4 as the current position, we construct the corresponding subsets of destinations based on the $new_des$ function. The set of destinations is divided to $D_{+x,\pm y,\pm z} = \{5, 8, 14, 26\}$, $D_{-x,\pm y,\pm z} = \{9, 15\}$, $D_{x,+y,+z} = \{24\}$. These 3 subsets express the number of replication in the node 5. In order to find the direction of replication, we partition the network based on these subsets of destinations as follows $N_{+x,\pm y,\pm z}$, $N_{-x,\pm y,\pm z}$ and $N_{x,+y,\pm z}$. These subnetworks produce the direction of replications which are east, west and north in our example.

Based on the result of the $replicate$ function for each pair of (current position, subdestinations), the $replicating$ event in the third model as shown in Fig. 20 is refined. Therefore, the nondeterministic assignment for the number of replication becomes equal to the cardinality of the $replicate$ function for the pair. Moreover, we refine the $routing$ event as given in Fig. 21 to a deterministic routing by selecting the output ports in the node based on the output of $replicate$ function.

```
Event replicating refines replicating
  any current_msg  pos
  where
    current_msg ↦ pos ∈ moving_pool
    replicate_flag(current_msg ↦ pos) = FALSE
    received_node⁻¹[{current_msg}] ∪ {pos} ⊂ des(current_msg)
  Witnesses
    num : num = card(replicate[{pos ↦ new_des(src(current_msg)
      ↦ pos ↦ des(current_msg))}])
  then
    replicate_flag(current_msg ↦ pos) := TRUE
    replicate_msg_num(current_msg ↦ pos) := card(replicate[{pos
      ↦ new_des(src(current_msg) ↦ pos ↦ des(current_msg))}])
  end
```

Fig. 20. $M_3$ : $MultiCast$: Replicating Event

```
Event routing refines routing
  where
    out_p = replicate(router ↦ new_des(src(msg) ↦ router ↦ des(msg)))
  then
    buffer_content := buffer_content ∪ ({msg} × ({router} × out_p))
  end
```

Fig. 21. $M_3$ : $MultiCast$: Routing Event

## V. Verification of the Models

Event-B employs a top-down refinement-based approach to system development. Development starts from an abstract system specification that models the most essential functional requirements. While capturing more detailed requirements, each refinement step typically introduces new events and variables into the abstract specification. These new events correspond to stuttering steps that are not visible at the abstract level.

The semantics of Event-B actions is defined using the so called before-after (BA) predicates [2], [5]. A before-after predicate describes a relationship between the system states before and after execution of an event. The semantics of a whole Event-B model is formulated as a number of *proof obligations*, expressed in the form of logical sequents. Below we present only most important proof obligations that should be verified (proved) for the initial and refined models. The full list of proof obligations can be found in [2].

The formal semantics provides us with a foundation for establishing the correctness of the Event-B specifications. In particular, to verify the correctness of a specification, we need to prove that its initialisation and all events preserve the invariant.

To verify the correctness of a refinement step, we need to prove a number of proof obligations for the refined model. For brevity, here we mention a few essential ones.

The initial Event-B model should satisfy the event feasibility and invariant preservation properties. For each event of the model, $evt_i$, its feasibility means that, whenever the event is enabled, its before-after predicate (BA) is well-defined, i.e., there exists some reachable after-state. Each event $evt_i$ of the initial Event-B model should also preserve the given model invariant. The event guards in the refined model can be only strengthened in a refinement step. The *simulation* proof obligation requires to show that the "execution" of the refined event is not contradictory with its abstract version.

The Event-B refinement process allows us to gradually introduce implementation details, while preserving functional correctness during stepwise model transformation. The model verification effort, in particular, automatic generation and proving of the required proof obligations, is significantly facilitated by the provided tool support – the Event-B platform.

The proof statistics for our models are shown in Table 1. These figures express the number of proof obligations generated by the Rodin platform as well as the number of obligations automatically discharged by the platform and those interactively proved. A high number of interactive proofs

were due to reasoning about set comprehension and unions, not currently supported automatically in Rodin. In addition, the interactive proving often involved manually suggesting values to discharging various properties containing logical disjunctions or existential quantifiers. Extra proving was due to the fact that currently, we cannot create proof scripts and reuse them whenever needed in RODIN. Thus, in some cases we had to manually repeat very similar or almost identical proofs.

TABLE I
PROOF STATISTICS

| Model | Number of Proof Obligations | Automatically Discharged | Interactively Discharged |
|---|---|---|---|
| Context | 21 | 6(28%) | 15(72%) |
| Multicast XYZ Context | 14 | 7(50%) | 7(50%) |
| $M_0$ Model | 121 | 77(64%) | 44(36%) |
| $M_1$ Model | 36 | 14(39%) | 22(61%) |
| $M_2$ Model | 34 | 8(24%) | 26(76%) |
| Unicast XYZ Model | 2 | 2(100%) | 0(0%) |
| Multicast XYZ Model | 5 | 2(40%) | 3(60%) |
| Total | 233 | 116(50%) | 117(50%) |

We note here two proof-related aspects. First, the interactive proofs are in a significant proportion rather easy to discharge. Second, the RODIN platform is continuously developing and thus, the number of interactive proofs is very likely to diminish in the near future. This will obviously increase the proof discharging process.

## VI. CONCLUSION

In this paper we have proposed a general model for the multicast communication in 3D NoCs. Our model is a generalization of a recently proposed model for unicast communication [12]; we show that the communication model in [12] is a special case or our more general model here. We apply the multicast communication model to develop two variants of the XYZ routing algorithm, one based on the unicast routing and the other on the tree-based routing. Both algorithms are thus provably correct.

Our general model for multicast communication can be further employed for analysis. For instance, we plan to compare different communication schemes in terms of efficiency and developing effort. Thus, we have a general platform for specifying, developing, and analyzing communication, based on provably correct modeling.

A side effect of modeling multicast communication is that we can employ it for modelling broadcast communication as well, by setting the destination set of a message to include all the network nodes in the ROUTER set. It remains to be seen if other broadcast methods, for instance flooding, can be generated directly from our general model or not.

## REFERENCES


[1] J. R. Abrial, *A system development process with Event-B and the Rodin platform*, ICFEM2007. LNCS, vol. 4789, pp. 1-3. Springer, 2007.
[2] J. R. Abrial, *Modeling in Event-B: System and Software Design*, Cambridge University Press, 2010.
[3] L.Tsiopoulos, M. Walden, Formal Development of NoC Systems in B, In *Nordic Journal of Computing*, pp. 127-145, 2006.
[4] J. R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.
[5] Rigorous Open Development Environment for Complex Systems (RODIN). *Deliverable D7, Event-B Language, online at http://rodin.cs.ncl.ac.uk/*.
[6] RODIN Tool Platform, http://www.event-b.org/platform.html.
[7] J. R. Abrial, D. Cansell, D. Mery, Refinement and Reachability in Even-B. In *4th International Conference of B and Z Users*, pp. 129-148, 2005.
[8] J. R. Abrial, S. Hallerstede, *Refinement, Decomposition and Instantiation of Discrete Models: Application to Event-B*, In *Fundamenta Informaticae*, pp. 1-28, 2007.
[9] R. J. Back and K. Sere. Stepwise Refinement of Action Systems. In *J. L. A. van de Snepscheut (ed), Proceedings of MPC'89 – Mathematics of Program Construction*, pp. 115-138, 1989.
[10] S. Katz, A Superimposition Control Construct for Distributed Systems. In *ACM Transactions on Programming Languages and Systems*, pp. 337-356, 1993.
[11] R. J. Back and K. Sere. Superposition Refinement of Reactive Systems. In *Formal Aspects of Computing*, Vol. 8, No. 3, pp. 324-346, Springer-Verlag, 1996.
[12] M. Kamali, L. Petre, K. Sere and M. Daneshtalab, *Refinement-Based Modeling of 3D NoCs*, In 4th IPM International Conferenece on Fundamentals of Software Engineering (FSEN11), to apear 2011.
[13] B. S. Feero and P. Pande, Networks-on-Chip in a Three-Dimensional Environment: A Performance Evaluation. In *IEEE Transactions on Computers*, pp. 32-45, 2009.
[14] I. Loi and L. Benini, An Efficient Distributed Memory Interface for Many-Core Platform with 3D Stacked DRAM. In *Proc. of the DATE Conference*, Germany, pp. 99-104, 2010.
[15] D. Park et al., Mira, A Multi-Layered On-Chip Interconnect Router Architecture, In *ISCA 2008*, pp. 251-261, 2008.
[16] A. Nayebi , S. Meraji , A. Shamaei , H. Sarbazi-azad, XMulator: A listener-Based Integrated Simulation Platform for Interconnection Networks, In *Asia International Conference on Modeling and Simulation*, pp. 128-132, 2007.
[17] M. Palesi, R. Holsmark, S. Kumar, V. Catania, Application Specific Routing Algorithms for Networks on Chip, In *IEEE transactions on Parallel and Distributed Systems*, pp. 316-330, 2009.
[18] R. Kaivola, R. Ghughal, N. Narasimhan, A. Telfer, J. Whittemore, et al, Replacing Testing with Formal Verification in Intel control sequence CoreTM i7 Processor Execution Engine Validation, In *Computer Aided Verification*, pp.414-429, 2009.
[19] J. Harrison, Formal Verification at Intel, In *Symposium on Logic in Computer Science*, 2003.
[20] R. Gupta, P. L. Guernic, S. K. Skuhla. *Formal methods and models for system design: a system level perspective*. Kluwer Academic Publishers, 2004.
[21] W. Liao , P. Hsiung, Creating a Formal Verification Platform for IBM CoreConnect-based SoC, In *the 1st International Workshop on Automasted Technology for Verificatin and Analysis (ATVA2003)*, pp. 7-18, 2003.
[22] E. A. Carara, F. G. Moraes, Deadlock-Free Multicast Routing Algorithm for Wormhole-Switched Mesh Networks-on-Chip, in Prof. of ISVLSI, pp.341-346, 2008
[23] M. Daneshtalab, M. Ebrahimi, S. Mohammadi, A. Afzali-Kusha, Low distance path-based multicast algorithm in NOCs, in IET Computers and Digital Techniques, Special issue on NoC, Vol. 3, Issue 5, pp. 430-442, Sep 2009.
[24] P. Mohapatra, V. Varavithya, A hardware multicast routing algorithm for two-dimensional meshes, in proc. Int. Conf. SPDP, New Orleans, 1996, pp. 198205.
[25] R. V. Boppana, S. C, C.S R, Resource deadlock and performance of wormhole multicast routing algorithms, IEEE Transactions on Parallel and Distributed Systems, pp. 535-549, 1998.
[26] P. Abad, V. Puente and J. . Gregorio, MRR: Enabling Fully Adaptive Multicast Routing for CMP Interconnection Networks, High Performance Computer Architecture (HPCA), 2009.

# Paper VII

## CorreComm: A Formal Hierarchical Framework for Communication Designs

**Maryam Kamali, Luigia Petre, Kaisa Sere and Masoud Daneshtalab**

Originally published in: *Proceedings of the 2nd IEEE International Conference on Networked Embedded Systems for Enterprise Applications - NESEA2011*, pp. 1-7. IEEE Computer Society, December, 2011.

# CorreComm: A Formal Hierarchical Framework for Communication Designs

Maryam Kamali*†, Luigia Petre*, Kaisa Sere* and Masoud Daneshtalab‡
*Åbo Akademi University, Finland
Email: {maryam.kamali,luigia.petre,kaisa.sere}@abo.fi
†Turku Centre for Computer Science (TUCS), Finland
‡University of Turku, Finland
Email: masdan@utu.fi

***Abstract*—The number of communicating components has tremendously increased, both at the chip-level communication as well as in general networks. This leads to an increased complexity in the design of communication infrastructures. In order to rely on such complex communication designs, we need a correspondingly increased verification effort. In this paper we propose a structured framework named CorreComm to alleviate the modeling and verifying of communication designs. We describe the correct-by-construction structure of our framework and demonstrate its applicability as a communication design pattern, by instantiating it to two specific communication models.**

***Index Terms*—Communication designs; Communication patterns; Formal methods; Event-B; Refinement**

## I. Introduction

Networks and communication are present nowadays in the everyday life, in an extremely pervasive manner. The financial systems, including banking and stock exchanges, the traveling systems, including booking flights and hotels, the electric grids administration, the nuclear plants processes, etc - are all examples of essential components of the society being gradually adapted to functioning online, via the internet or other networking configurations. At the chip level, the Network-on-Chip (NoC) communication paradigm [1], based on a modular packet-switching mechanism, can address many of the on-chip communication design issues such as performance limitations of long interconnects and the integration of high numbers of Intellectual Property (IP) cores on a chip. Multicore chips are embedded in a multitude of systems and appliances, effectively transforming the way we use our phones and TVs, the way we pay for our shopping, the way we travel, and the way our health is supported, to only name a few applications. The communication designs for this variety of systems has gradually become more and more complex and more and more essential for us.

Given the widespread and relevance of communication, it is imperiously necessary to be able to rely on the communication designs, i.e., to be certain of their various properties. Formal methods, with their mathematic proving core, are an important instrument in ensuring the integrity of software-intensive systems. Traditionally characterized as hard to use, due to the requested mathematical background and the lack of automatic tools, nowadays formal methods have matured, to the point where they are considered in industry when developing software-intensive systems [2]. Examples of the industrial undertaking of formal methods are increasing. The famous line 14 of the driverless Parisian metro [3], developed in 1998 using the B-method [4], is the first notable example of a formal method-based development, reviewed in [5]. The method used by Siemens for developing the software controlling the line 14 train ensured its correctness in a mathematical manner that effectively eliminated the unit testing from the software lifecycle. No human resources are now needed to operate the trains and in addition, the trains are faster, hence fewer are needed in total.

More recent examples of the Event-B [6] formal method usage in industry can be seen for instance with Space Systems [7] and SAP [8]. In Event-B, the development of a model is carried out step by step from an abstract specification to more concrete implementations. Using the *refinement* approach, a system can be described at different levels of abstraction, and the consistency in and between levels can be proved mathematically.

In this paper we propose a hierarchical framework named CorreComm for modeling communication designs as well as for verifying their correct construction. Modeling and verifying full communication designs is very complex and time-consuming. In our methodology we propose to model and verify different components of the communication designs in a stepwise and hierarchical manner. We ensure the correctness of these components by developing them via refinement in Event-B. Moreover, these components can be further refined in order to model various specific communication design choices. Hence, the contribution of our paper can be summarized as follows:

- we propose CorreComm, a framework for facilitating the design exploration of possible communication infrastructures
- we introduce a methodology for alleviating the verification effort for communication designs
- we demonstrate the applicability of the methodology by deriving two specific communication models, namely the first-in, first-out (FIFO) buffer and the wormhole switching technique

978-1-4673-0498-6/11/$26.00 ©2011 IEEE

We proceed as follows. In Section II we describe Event-B to the extent needed in this paper. In Section III we describe the main components of CorreComm and in Section IV we detail the stepwise construction of CorreComm in Event-B. In Section V we discuss the applicability of CorreComm as a communication design pattern and in Section VI we bring forward the verification techniques employed in this paper. Related work is briefly surveyed in Section VII. We conclude the paper in Section VIII.

## II. The Event-B Modeling Method

Event-B [6] is a formal method for modeling and reasoning about distributed systems. The semantics of Event-B is based on transition systems and before-after predicates. Modeling in Event-B is based on set theory and first order logic, while the key feature of Event-B consists in promoting the correct-by-construction approach based on the refinement concept [9], [10], [11]. This means that a system can be modeled in various degrees of detail at different abstraction levels; the model consistency of each level can be verified, together with the consistency of the models in between levels. The refinement approach promotes the development of a model gradually, by making it more precise and detailed in each step. This leads to the control of complexity as system development is based on a model chain, with each model a refinement of the previous models in the chain.

Event-B models are composed of *contexts* and *machines*. A context describes the static part of a model, made of *carrier sets* and *constants*, together with *axioms* describing the properties of these. A machine describes the dynamic part of a model. Machines can contain *variables, invariants* and *events*. The values of the variables $v$ define the state of a machine. Variables are constrained by invariants $I(v)$ which should be preserved by any changes in the value of the variables. The state changes are specified by a number of *events*. Each event is formed of a *guard* and an *action*. The guard $G(t, v)$, where $t$ are the local parameters of the event, is the necessary condition under which the event may occur. The action $S(v, t)$ models how the state variables change when the event executes. An event may be executed only when its guard holds. The semantics of Event-B actions is defined using the so called before-after (BA) predicates [6], [12]. A before-after predicate describes the relationship between the system states, before and after the execution of an event.

A refined model $RM$ usually has more variables than its abstraction $AM$. The state of $AM$ is related to the state of the $RM$ by a gluing invariant $J(v, w)$, where $v$ are the variables of $AM$ and $w$ are the variables of $AM$. There are several types of refinement, out of which we employ in this paper superposition refinement and data refinement. Superposition refinement [13], [14] corresponds to adding new data and new events on top of the already existing data and events, but in such a way that the introduced behavior does not contradict or take over the abstract machine behavior. Data refinement [15] refers to replacing some (more abstract) variables with some other (more concrete) variables in a manner controlled by the gluing invariant. Correspondingly, the events dealing with the old variables are replaced by several corresponding events in the refined machine.

In this paper we describe all the Event-B modeling in the form of simplified explanations and diagrams, due to lack of space. However, the discussed models are fully developed and proved in Event-B and will appear as online material in due time.

## III. The Structure of CorreComm

In this section we introduce the structure of the CorreComm framework and outline the purpose of each component of the framework.

With CorreComm we propose that a communication design consists of three main components: `Communication Architecture`, `Communication Primitives`, and `Communication Properties`, as illustrated in Fig. 1.

The `Communication Architecture` component describes the communicating elements, their relationship to each other, and various other constructs needed for communication. It consists of two sub-components as illustrated in Fig. 1: a `Structural Elements` sub-component and a `Relationships` sub-component. Further on, there are three main types of structural elements involved in communication designs, namely `Node`, `Channel` and `Buffer`. A node denotes any type of communicating unit. A channel connects nodes together. A buffer stores data temporarily in the start and end points of the channels. The behavior of each individual element can be varied from a communication design to another. In other words, the basic functionality of each element can be extended or refined to a specific design scheme. However, this foundation is common to all the communication architectures. The relationships between the elements in the network shows the topology of the architecture.

The communication architecture shows how different elements are connected to each other in order to transfer messages. The message transfer methods are described in the `Communication Primitives` component of our framework. There are four main communication primitives, namely `Injecting`, `Switching`, `Routing` and `Receiving`, as illustrated in Fig. 1. Injecting refers to introducing a new message in the network. Routing refers to deciding the route that a message should follow. Switching determines when the routing decisions are made, the setting/resetting of the switches inside the nodes, and how the messages are transferred inside the nodes. Receiving is the function that deletes the messages having reached their destination from the network.

The `Communication Properties` component is intended for the modeling and verification of the communication design properties, based on the given architecture and primitives. The main purpose of formal modeling a communication design is to verify the correctness of communication. A typical property for the functional correctness of communication is that all the injected messages are received by all their destinations. However, other properties can be modeled as well, for instance non-functional properties such as the performance

978-1-4673-0498-6/11/$26.00 ©2011 IEEE

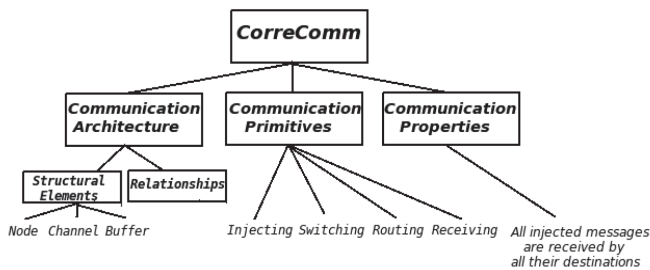



Fig. 1. Communication Scheme

or energy consumption of a given communication design. In this paper we only refer to the functional correctness of communication.

## IV. The stepwise construction of CorreComm in Event-B

In this section, we describe the modeling of the proposed structure of communication designs in Event-B and discuss its correctness. The communication architecture describes the static part of a communication network, so all its elements are specified in the context part of the model as sets, constants, and axioms. The communication primitives define the dynamic behavior of the communication network, therefore are modeled in the machine part of the Event-B model as variables and events. The communication properties are modeled as a list of invariants in the machine part of the model and verified by a list of proof obligations. In the following we outline this development.

We model the communication design elements in a stepwise manner, employing the abstraction and refinement concepts put forward in Section II. For every abstraction level we have a corresponding context, machine, list of invariants, and list of proof obligations to discharge. The initial model consists of the least number of structural elements so that the correctness property is formulated. In the subsequent levels, we gradually add the other structural elements in such a way that they do not contradict with the previous models. As we hierarchically refine the architecture, the communication primitives component is also hierarchically refined. In each abstraction level we verify the communication properties, expressed in terms of the (hierarchically-)corresponding architectural elements and primitives. The stepwise development continues until all the elements of the communication design are modeled. The resulted model can then be employed to specify more precise structural elements as well as specific communication designs.

In Level 1 of abstraction, we consider just a small number of structural and primitive elements, essentially as few as possible for expressing relevant invariants. This means that some elements are abstracted away from this initial model. More precisely, in Level 1 we model the structural element `Node` by defining a set *Node* that models the elements of the communication network. Technically, the set of nodes is a parameter of our model. The relationships between these nodes, defining the network topology, are modeled with a constant named *Neigh*. This constant effectively records the pairs of related nodes. In order to model the functional correctness of the communication, we need to specify that all the injected messages are received by all their destinations. For this, we model another set, $Msg$, as a parameter of our modeling. A message in the network has a source node and a set of destination nodes, hence any element of $Msg$ is a triple $(msgID, src, \{des\})$.

We model the corresponding communication primitives in Level 1 with variables and events. Thus, in the initial model we have four abstract variables: $snt_pl, mv_pl, rcv_nd$ and $rcv_pl$. These variables are enough for modeling (and verifying) the functional correctness of the communication. The $snt_pl$ variable denotes the list of messages which are injected by a source node to the network. The $mv_pl$ variable denotes the current position of messages still traveling in the network. The $rcv_nd$ variable models the instances of messages having reached destinations. When a message is received to one of its destinations, a corresponding map is added to the $rcv_nd$ relation. The $rcv_pl$ denotes the list of messages which are removed from the network by all their destinations, i.e., a message-to-destination map for all of the destinations of that message should be in $rcv_nd$.

The four communication primitives are abstractly modeled in Level 1 by three events: $inject_msg$, $travel_msg$, and $receive_msg$. The $inject_msg$ event handles the injecting primitive by adding a new message to the $snt_pl$ set and the $mv_pl$ set. The reason for adding the message to $mv_pl$ is to launch the message transferring process. The $travel_msg$ event handles both switching and routing primitives in Level 1, by modeling the transferring of a message from a node to another. We thus postpone the modeling of specific routing and switching primitives for the subsequent refinement steps. The $receive_msg$ event handles the receiving primitive of the communication model in two different situations. Either the message is received by all its destinations or, there are still some destinations that did not receive the message. In the former case, the message is added to the $rcv_pl$ and removed from the $mv_pl$. In the latter, the corresponding message-to-destination map is added to $rcv_nd$ and the message is not added to $rcv_pl$ because there are still destinations waiting for the message.

We model the functional correctness of the communication, i.e., that all the injected messages are received by all their destinations, with the following invariant:

$$mv_pl = \varnothing \Leftrightarrow snt_pl = rcv_pl$$

This predicate models the equivalence between no traveling message (i.e., all the traveling messages have reached their destinations) and the receiving pool of messages containing all the injected messages.

The initial model forms the first hierarchical level, Level 1, of our development. Level 1 is a foundation model for the subsequent refinement steps, where we refine the context part

978-1-4673-0498-6/11/$26.00 ©2011 IEEE

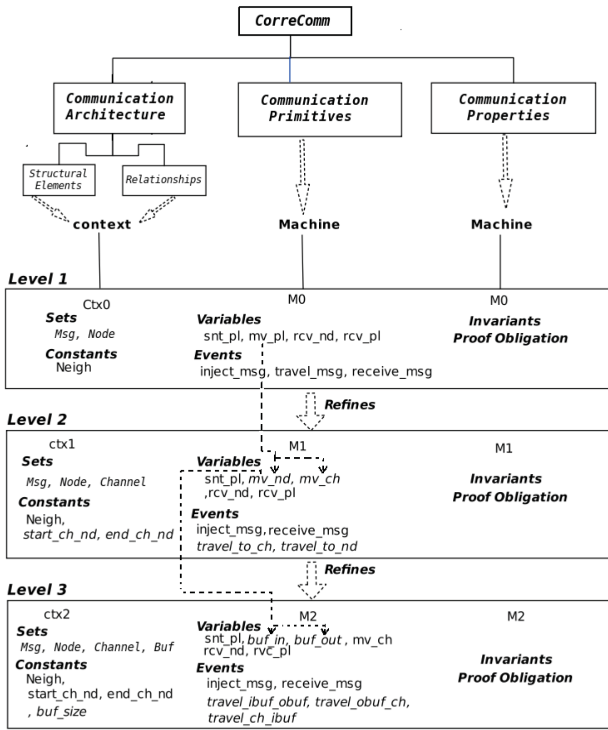



Fig. 2. Communication Scheme in Event-B

of the model to extend the communication architecture and refine the machine part of the model to refine the communication primitives. The mapping between the communication design to Event-B is summarized in Fig. 2.

In Level 2, we extend the initial context to model the `Channel` structural element. We specify the set $Channel$ that (physically) connects neighbor nodes to each other. In other words, for any pair in the $Neigh$ relation, we model a channel that connects the pair by modeling two relations: $start_ch_nd : Node \leftrightarrow Channel$ and $end_ch_nd : Channel \leftrightarrow Node$. The composition of these two relations is equal to $Neigh$ relation. Now, we refine the communication primitive component to specify the transferring of messages through the channels. In this level of hierarchy, we use the data refinement technique. In the initial model, the $mv_pl$ relation denotes the current position of messages in the network. In order to represent the existence of messages in both nodes and channels, in the second model we replace the $mv_pl$ variable with two variables: $mv_nd$ and $mv_ch$. The $mv_nd$ variable denotes the current position of those messages that are in nodes and the $mv_ch$ variable denotes the current position of those messages that are in channels. In fact, the union of messages in these two relations is equal to those messages in $mv_pl$.

$$mv_pl = mv_nd \cup (mv_ch; start_ch_nd)$$

As a consequence of this substitution, the events that represent the communication primitive component are updated, so that all the connections to $mv_pl$ are substituted either by $mv_nd$ or by $mv_ch$. In addition, we refine the $travel_msg$ event by splitting it into two events: $travel_to_ch$ and $travel_to_nd$. The $travel_to_ch$ event models the transferring of a message from a node to a channel and the $travel_to_nd$ models the transferring of a message from a channel to a node at the end of the channel. This data refinement comes with a list of proof obligations that express the correctness of the replacement. These obligations guarantee that the second model preserves the communication properties component.

In the third model (Level 3) we add the `Buffer` structural element by modeling a parameter set $Buf$ and a constant $Buf_size$. We refine accordingly the dynamic part of the model. Intuitively, at the start and at the end points of any channel, we now model a buffer of size $Buf_size$ that stores the messages. Each node consists of a number of input and output buffers depending on its input and output channels. Any message in a node settles in either an input buffer of the node or an output buffer of the node. Therefore, the $mv_nd$ variable is replaced by the $buf_in$ and $buf_out$ variables, via data refinement. The $buf_in$ variable denotes the content of the input buffers of nodes and the $buf_out$ variable denotes the content of the output buffers for nodes.

At this level of abstraction, we refine the $travel_to_nd$ event in the second model to the $travel_ch_ibuf$ event. The $travel_ch_ibuf$ event models the receipt of a message from a channel, provided that the buffer at the end of the channel has space to add a new message. This event releases the channel and adds the message to the input buffer of the node corresponding to the end of the channel. The $travel_to_ch$ event that models transferring of a message from a node to some neighbors is refined to the $travel_obuf_ch$ event. When there is a message in an output buffer and the channel corresponding to the buffer is free, the $travel_obuf_ch$ removes the message from the output buffer and adds it to the channel. The newly introduced $travel_ibuf_obuf$ event models the transfer of a message from an input buffer of a node to a number of output buffers of the node in a nondeterministic manner. This nondeterminism can be refined to specific routing techniques in subsequent refinements.

In this refinement step we establish the connection between abstract variables and the more concrete variables by the following gluing invariant:

$$\begin{aligned}&\forall msg, n.msg \mapsto n \in mv_nd \Leftrightarrow \\ &\quad (\exists ch.(ch \in start_ch_nd^{\sim}([\{ch\}]) \wedge msg \in buf_in(ch)) \\ &\quad \vee (ch \in end_ch_nd^{\sim}([\{ch\}]) \wedge msg \in buf_out(ch)))\end{aligned}$$

This invariant guarantees that all the mappings in the $mv_nd$ relation in the abstract model are in either $buf_in$ or $buf_out$. When we prove that the invariant preserve by all the events in the machine, we guarantee the communication correctness.

The third level of hierarchy presents the foundation framework of the communication design. The way which is developed shows how we can use the framework and the refinement method to model and verify a specific communication designs.

978-1-4673-0498-6/11/$26.00 ©2011 IEEE

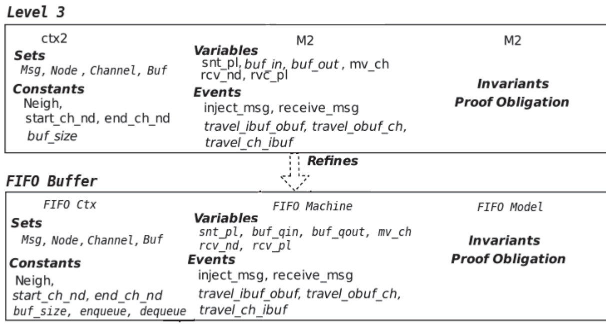


Fig. 3. FIFO Buffer Model

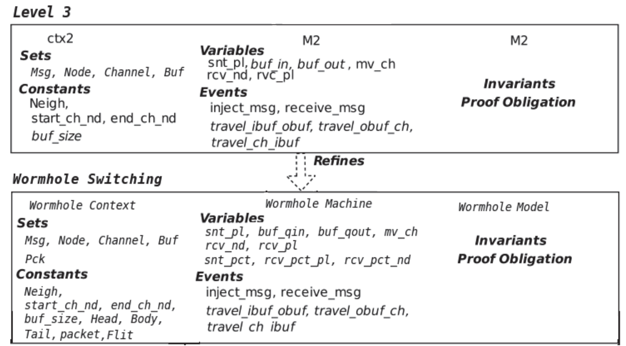


Fig. 4. Wormhole Model

As it is shown, in each hierarchical level we gradually add more detail to the abstract one until to acheive a complete model of the system. This method let us understand the problem entirely and also focus to a part of the system at each level.

As a case study, we refine the buffer structural element to a FIFO-buffer and we refine the communication function to model the wormhole switching technique.

## V. Instantiating CorreComm

To demonstrate that CorreComm can be used as a communication design pattern, we have employed it for modeling a first-in, first-out (FIFO) buffer as well as the wormhole switching technique. In the following we summarize our modeling.

### A. The FIFO buffer

A FIFO buffer is simply a buffer with specific rules for element addition and removal. In order to model a FIFO buffer we extend the static part of CorreComm, where the communication structural elements are defined. We introduce a queue data structure with two functions, $enqueue$ and $dequeue$, in the context part of the model. This data structure models a FIFO buffer. In the machine part of the model we substitute the $buf_in$ and $buf_out$ variables with two new concrete variables $buf_fifo_in$ and $buf_fifo_out$. Correspondingly, we refine all the events for replacing the previous buffer specification with the new one as illustrated in Fig. 3. We also prove that the new buffer structure is a refinement of the basic buffer specification given in the Level 3 of CorreComm.

We observe that any buffer structure can be modeled in this refinement step, by suitably defining the $enqueue$ and $dequeue$ functions. It is sufficient to model different buffer structures in different contexts and use any of them in the machine part of the model. Thus, our modeling methodology provides a framework for producing a list of libraries for specifying communication architectures.

### B. The wormhole switching technique

In Levels 1-3, we do not take the order of the traveling messages into consideration, but only prove that all the messages are received to their destinations. This means that messages belonging to a packet can be received to their destinations in any order. In the wormhole switching technique, messages belong to a packet travel and are received to their destinations in a certain order. Every packet is divided into parts called *flits*. Flits are categorized into three types: *head, body* and *tail*. A head flit models the head of a worm traveling in the network. The body flits follow the header flit and the tail flit follows the body flits, denoting the end of the worm.

To model the wormhole switching, we refine the previous context model to assign flit types to messages of a packet as illustrated in Fig. 4. For this we model a total function from $Msg$ to a set of flit types $\{Head, Body, Tail\}$ ($Flit : Msg \rightarrow \{Head, Body, Tail\}$). In addition, the relation between messages and their packets is modeled as a total surjective relation from $Msg$ to their packets ($packet : Pck \twoheadleftrightarrow Msg$). This relation models that each message can be part of only one packet and a packet is formed by a unique subset of messages. We assume that each packet consists of a head, a tail and at least one body flit.

Based on the static part of the model, we extend the dynamic part to describe the traveling and receiving of packets and messages by considering the wormhole switching technique. We define three new variables: $snt_pct_pl$, $rcv_pct_pl$ and $rcv_pct_nd$. The $snt_pct_pl$ variable denotes the list of packets injected by a source node to the network. When all flits of a packet are added to the network, the packet is also added to the $snt_pct_pl$. When a tail flit of a packet is added to the network it means that all the flits of the packet have been already added to the network. We model this property as an invariant. The $rcv_pct_nd$ variable models the instances of packets having reached destinations as a set of maps. When a tail flit of a packet is received to a destination, it means that all the flits of the packet have been already received. The $rcv_pct_pl$ variable denotes the list of packets which are removed from the network by all their destinations. These properties are modeled as the following invariants.

$$\forall p, m. m \in snt_pl \land m \in packet[\{p\}] \land Flit(m) = Tail \Leftrightarrow p \in snt_pct_pl$$

$$\forall p, m. m \in rcv_pl \land m \in packet[\{p\}] \land Flit(m) = Tail \Leftrightarrow p \in rcv_pct_pl$$

In addition, we refine the events to modify the values of the

978-1-4673-0498-6/11/$26.00 ©2011 IEEE

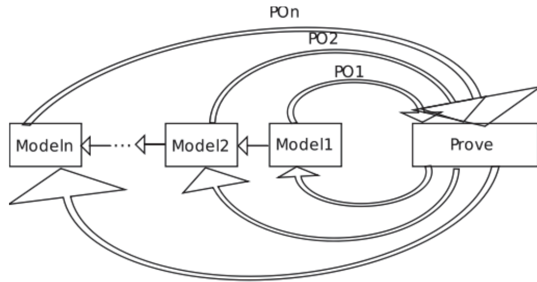



Fig. 5. Formal Development in Event-B

new variables. Thus, we refine the $inject_msg$ event to add a packet to $snt_pct_pl$ if the current injected message is a tail. We also add new constraints in the guard of the event to model the order of the injected messages to the network. The guards and actions of other events are similarly extended to model the traveling and receiving of packets based on the flow control defined by the wormhole switching technique. More precisely, the head of a packet is always traveling in the network with the following subsequent flits and there are no flits from other packets between them. These properties are given as a list of invariants in the model to guarantee the correctness of the model.

## VI. ON THE VERIFICATION OF CORRECOMM

The main purpose of formal modeling consists in reasoning about and finding out the design problems. In the Event-B framework reasoning is supported by formal proof. With each formal model we generate a number of *proof obligations* that establish the properties of our model. The proof obligations, described in the form of logical sequents, express the semantics of a whole Event-B model.

In this paper, we rely on the Rodin Platform [16], [12] tool where proof obligations are generated automatically and proven either automatically or interactively. The Rodin platform [12], which is based on Eclipse, facilitates modeling in the Event-B language. In addition to providing a user interface for editing Event-B models, it generates proof obligations regarding the model on the fly, updating these every time the model is saved, discharges some of these proof obligations automatically and allows the user to interactively discharge the rest.

By discharging the proof obligations we guarantee the correctness of our model. However, when we fail to discharge some proof obligations, we modify the model and again try to discharge the proof obligations of the modified model. The modification process continues until all proof obligations are discharged. In the Event-B formalism, discharging proof obligations thus provides a means to deeply understand and reason about our designs. Constructing a correct model of a system in Event-B is a cycle of modeling and proving proof obligations in different level of abstractions as shown in Fig. 5.

To exemplify the cycle of modeling and proving, we consider one of the behavioral properties of Event-B models, namely *invariant preservation*. This means that we need to verify if invariants hold whenever variables change their values via execution of events. While modeling CorreComm, we observed that invariant preservation properties for the $receive_msg$ in the initial model (Level 1) were not automatically discharged. By interactive discharging we found a failure in the guard of the event. In the $receive_msg$ event, when a message is received to all its destinations, it is removed from the $mv_pl$ variable and added to the $rcv_pl$ variable; however, when it is not received to all its destinations, no update is done for the $mv_pl$ and $rcv_pl$ variables. In the faulty event, we had considered both possible actions while only one of the above conditions had been included in the guard. Upon modifying the guard of event, we finally could discharge the invariant preservation property for the $receive_msg$ event.

The summary of discharged proof obligations in the final version of CorreComm is displayed in Table 1.

TABLE I
PROOF STATISTICS

| Model | Number of Proof Obligations | Automatically Discharged | Interactively Discharged |
|---|---|---|---|
| $Level1$ Context | 2 | 2(100%) | 0(0%) |
| $Level2$ Context | 1 | 1(100%) | 0(0%) |
| $Level3$ Context | 0 | 0(0%) | 0(0%) |
| $FIFO$ Context | 15 | 15(100%) | 0(0%) |
| $Wormhole$ Context | 2 | 2(100%) | 0(0%) |
| $Level1$ Machine | 39 | 32(82%) | 7(18%) |
| $Level2$ Machine | 67 | 47(70%) | 20(30%) |
| $Level3$ Machine | 120 | 80(67%) | 40(33%) |
| $FIFO$ Machine | 52 | 42(81%) | 10(19%) |
| $Wormhole$ Machine | 138 | 95(69%) | 43(31%) |
| Total | 436 | 316(72%) | 120(28%) |

## VII. RELATED WORK

Formal methods have been used before for modeling and verifying communication designs at different levels of abstraction and in different application domains. For instance, in [17], [18] specific communication protocols at the network layer are modeled and verified independently of the underlying infrastructure. In [19], the authors specify and verify the communication correctness of the network infrastructure in a system-level model.

In [20], a framework for designing the communication network is proposed. Authors present a model for communication synthesis. The model consists of quantities to measure the performance of a communication component, composition rules to handle how to build composite components, and communication structures to capture the behavior of composite components. In addition, in [21], a modeling framework for communication architectures is proposed. The authors present an application-driven platform for object-oriented modeling of the communication architectures. They introduce a specific hierarchical class library that is used to develop new communication architectures. However, neither [20] nor [21] propose any verification methods for their communication designs.

Summarizing, there are a number of informal platforms for the modeling and evaluation of communication designs.

Besides, there are mature formal techniques that could be employed for modeling, verifying, and reasoning about the communication designs. However, the formal techniques have not been applied in the platform level and instead, the majority of existing formal approaches focus on particular networking and communication aspects. Our main contribution in this paper is to introduce CorreComm as a formal hierarchical framework for modeling and verifying communication designs in order to bridge this gap. For instance, in [22], [23], [24], formal models are introduced to study unicast and multicast communication at the chip level as well as to study recovery mechanisms in wireless sensor-actor networks, respectively. With CorreCom, we establish a base communication design, out of which the formal models mentioned above could be derived via refinement in a correct-by-construction manner.

## VIII. Conclusions

In this paper we propose CorreComm, a formal hierarchical framework for modeling and verifying communication designs. CorreComm is developed stepwise based on the Event-B formal method, so that we are sure of its consistency. We have also demonstrated the value of CorreComm as a communication design pattern, in that in can be reused, extended, and refined whenever a communication design needs to be elaborated. With CorreComm we thus avoid errors in the early stages of elaborating communication designs. Also, when extending CorreComm, one can detect errors in the early stages of construction, due to the interactive style of modeling and proving in Event-B via the Rodin platform.

CorreComm can be used to construct a library of verified components to be employed for the verification of specific communication designs. We plan to expand the library of verified components by continuing to model various switching and routing techniques. In addition, we plan to focus on modeling power consumption issues as communication properties and, in general, to explore the quantitative analysis of the communication designs within our formal hierarchical framework. This can easily be seen as extending the `Communication Properties` component of CorreComm. This would allow us to evaluate different parameters of the communication designs and, as a consequence, come up with optimal solutions for them.

## References


[1] M. Daneshtalab, M. Ebrahimi, P. Liljeberg, Juha Plosila, and H. Tenhunen, A Low-Latency and Memory-Efficient On-chip Network, In proceedings of 4th IEEE/ACM International Symposium on Network-on-Chip (NOCS), pp. 99-106, May 2010.

[2] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald. Formal Methods: Practice and Experience. In *ACM Computing Surveys*, Vol. 41, Issue 4, pp. 1-36, 2009.

[3] D. Craigen, S. Gerhart and T.Ralson, Case Study: Paris Metro Signaling System, In *the Proceedings of IEEE Software*, pp. 32-35, 1994.

[4] J. R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[5] T. Lecomte. Applying a Formal Method in Industry: A 15-Year Trajectory. In *M. Alpuente, B. Cook and C. Joubert (Eds), Proceedings of the Formal Methods for Industrial Critical Systems, FMICS 2009*. Lecture Notes in Computer Science, Vol. 5825, pp. 26-34, Springer-Verlag, 2009.

[6] J. R. Abrial, *Modeling in Event-B: System and Software Design*, Cambridge University Press, 2010.

[7] A. S. Fathabadi, A. Rezazadeh, and M. Butler. Applying Atomicity and Model Decomposition to a Space Craft System in Event-B. In: *Proceedings of the THIRD NASA FORMAL METHODS SYMPOSIUM*, Lecture Notes in Computer Science, Vol. 6671, pp. 328-342, 2011.

[8] J. Bryans and W. Wei. Formal Analysis of BPMN Models Using Event-B. In the *S. Kowalewski and M. Roveri (Eds.), Proceedings of the 15th International Workshop on Formal Methods for Industrial Critical Systems, FMICS 2010*. Lecture Notes in Computer Science, Vol. 6371, pp. 33-49, Springer-Verlag, 2010.

[9] R. J. Back and J von Wright. *Refinement Calculus: A Systematic Introduction*. Springer-Verlag, 1998.

[10] J. R. Abrial, D. Cansell, D. Mery, Refinement and Reachability in Even-B. In *4th International Conference of B and Z Users*, pp. 129-148, 2005.

[11] J. R. Abrial, S. Hallerstede, *Refinement, Decomposition and Instantiation of Discrete Models: Application to Event-B*, In *Fundamenta Informaticae*, pp. 1-28, 2007.

[12] Rigorous Open Development Environment for Complex Systems (RODIN). *Deliverable D7, Event-B Language, online at http://rodin.cs.ncl.ac.uk/*.

[13] S. Katz, A Superimposition Control Construct for Distributed Systems. In *ACM Transactions on Programming Languages and Systems*, pp. 337-356, 1993.

[14] R. J. Back and K. Sere. Superposition Refinement of Reactive Systems. In *Formal Aspects of Computing*, Vol. 8, No. 3, pp. 324-346, Springer-Verlag, 1996.

[15] R. J. Back and K. Sere. Stepwise Refinement of Action Systems. In *J. L. A. van de Snepscheut (ed), Proceedings of MPC'89 – Mathematics of Program Construction*, pp. 115-138, 1989.

[16] RODIN Tool Platform, http://www.event-b.org/platform.html.

[17] D. Borrione, A. Helmy, L. Pierre, and 1. Schmaltz. A Formal Approach to the Verification of Networks on Chip. *EURASIP Journal on Embedded Systems*, 2009(548324):1-14, February 2009.

[18] G. Salaun, W. Serwe, Y. Thonnart, and P. Vivet. Formal Verification of CHP Specifications with CADP Illustration on an Asynchronous Network-on-Chip. In Proceedings of International Symposium on Asynchronous Circuits and Systems, pages 73-82. IEEE Computer Society Press, March 2007.

[19] Y. Chen, W. SU, P. Hsiung, Y. Lan, Y. Hu, and S. Chen, Formal Modeling and Verification for Network-on-Chip, In Proceeding of International Conference on Green Circuits and Systems (ICGCS), pp. 299-304, 2010.

[20] A. Pinto, A. Sangiovanni-Vincentelli and L. P. Carloni, COSI: A Framework for the Design of Interconnection Networks, *Design and Test of Computers*, IEEE, Vol. 25 Issue:5, pp. 402-415, 2008.

[21] X. Zhu and Sharad Malik, A hierarchical modeling framework for on-chip communication architectures of multiprocessing SoCs, *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, Vol. 12 Issue 1, 2007.

[22] M. Kamali, L. Petre, K. Sere and M. Daneshtalab, *Refinement-Based Modeling of 3D NoCs*, In 4th IPM International Conferenece on Fundamentals of Software Engineering (FSEN11), to apear 2011.

[23] M. Kamali, L. Petre, K. Sere, M. Daneshtalab, Formal Modeling of Multicast Communication in 3D NoCs, In *the 14th EUROMICRO Conference on Digital System Design (DSD)*, appear 2011.

[24] M. Kamali, L. Laibinis, L. Petre, K. Sere, Self-Recovering Sensor-Actor Networks, In proceedings of 9th International Workshop on the Foundations of Coordination Languages and Software Architectures, FOCLASA2010, Sep 2010.


978-1-4673-0498-6/11/$26.00 ©2011 IEEE

# Paper VIII

## NetCorre: A Hierarchical Framework and Theory for Network Design

**Maryam Kamali, Luigia Petre and Kaisa Sere**

*Submitted to Science of Computer Programming*

# NetCorre: A Hierarchical Framework and Theory for Network Design

Maryam Kamali[a,b], Luigia Petre[a], Kaisa Sere[a]

[a] *Department of Information Technologies, Åbo Akademi University, Turku, Finland*
[b] *Turku Centre for Computer Science (TUCS), Turku, Finland*

**Abstract**

Design decisions made at the algorithm level affect the implementation significantly. In this paper, we introduce a formal framework for high-level analysis of network design in which both design aspects of algorithm and architecture are integrated. We hierarchically develop the framework model relying on the formal design technique of refinement. In addition, we extract the common features of networks as a library of design rules that are reusable and extendable for modelling and analysing different network designs. We elaborate our framework model within the Event-B formalism, using the Rodin platform - an integrated development framework for Event-B; in particular, for modelling the library of design rules, we employ the novel theory extension mechanism of Event-B and the corresponding plug-in of the Rodin platform.

*Keywords:* Algorithm-architecture network design; Formal hierarchical framework; Refinement; Event-B; RODIN Tool

## 1. Introduction

Design decisions made at the algorithm level affect the implementation significantly. Sometimes it is not even possible to apply some algorithms to some network architectures, and even when this is possible, many time-consuming optimization techniques might still be needed. It is thus important to be able to analyse the trade-offs of various algorithm-architecture alternatives at higher levels, before investing time and effort in exploring them at lower levels. For this, both design aspects of algorithm and architecture need to be integrated in a framework for high-level analysis, to allow for a detailed assessment of implementation parameters.

Some integrated design frameworks have already proven successful. A design exploration methodology is developed in [17] for wireless communications. The author adapts a top-down approach to design optimization with a bottom-up module-based approach for implementation estimates. A design platform for algorithm-architecture design exploration for efficient implementation of multimedia algorithms is proposed in [16]: a pure software specification is gradually transformed into a mixed software and hardware implementation.

However, even if the algorithm and the architecture are integrated in these methodologies, a significant number of errors as well as undesirable behaviour have been identified. This strongly motivates the need for checking the integrity of the proposed network designs. An increasingly viable approach for facilitating the design of correct systems is provided by formal methods. Formal methods are dedicated to the problem of system validation, being able affect the entire lifecycle of system development from specification to implementation. System validation ensures that a system satisfies its design specification and can be performed via testing, simulation, and verification. Verification consists of logical reasoning on the system specification and thus can detect and avoid possible design errors before system implementation. Validation performed via testing and simulation is limited to the test cases, while verification allows the consideration of all system behaviours.

Therefore, it is clear that formal specification and verification of network designs would increase their reliability and lower their implementation cost. This is mainly due to error detection in the early stages of design. However, using formalisms has some challenges, such as unambiguously formulating properties of algorithms and architectures as well as proving them. Considerable amounts of expressiveness as well as experience typically help in deriving models and their desired properties.

In this context, we aim at proposing a high-level formal framework that integrates the algorithm and architecture analysis for network design. Moreover, our proposed framework addresses the challenge of applying formal methods in the early stages of design. In this paper, with *network design* we refer to the integrated algorithm-architecture approach to networking and with *modelling the framework* we refer to the formal specification and verification of this integrated framework. The two central features of our proposed framework are that (1) it is intended for reuse in designing verified networks and (2) it is hierarchical; the latter feature is instrumental both in developing the framework and in its intended reuse.

The contribution of this paper is twofold. First, we introduce a reusable, hierarchical formal model of this framework that is verified. Second, from the hierarchical model we extract a reusable *network theory*, also pre-proven. The formal model and the theory are together referred to as NetCorre, for correct networking. We base our work on the Event-B formal method [3, 1] for system modelling and analysis. The refinement technique of Event-B is fundamental in our approach because it allows to represent a system at different abstraction levels and to prove the consistency between these. Event-B comes with an associated toolset Rodin, a theorem prover-based environment where proofs about the models are generated automatically and discharged either automatically or interactively. Importantly, the Rodin platform has extension capabilities: one can define new theories consisting of new data types, operators on them, etc. These new theories can be proven in Rodin and then reused as original Event-B constructs. Our proposed network theory falls exactly within this reuse pattern. Hence, upon defining a reusable, hierarchical formal model for network design, we extract the fundamental components to be reused and define the NetCorre network theory. In a nutshell, our contribution in this paper can be summarized as *correct network reuse*.

We proceed as follows. In Section 2 we review related approaches of informal and formal network design. In Section 3, we introduce the Event-B formal method that we use, together with its extension mechanisms. In Section 4 we outline the hierarchical framework of algorithm-architecture network design. In Section 5, we detail the development of the framework, based on the refinement technique. In Section 6, we put forward our network theory proposal and then we wrap up in Section 7.

## 2. Related Work

An algorithm-architecture design methodology for wireless communications is proposed in [17]. Algorithms are evaluated at an early stage based on extracting critical characteristics. Their mappings to various architectures are then identified in order to improve implementation efficiency. The purpose of the proposed methodology in [17] is to optimize the high-level design by analysing design trade-offs.

Another approach for exploring an integrated algorithm and architecture design is presented in [16]. A platform is proposed to evaluate the efficiency of multimedia algorithms in hardware implementations. In contrast to these

two design approaches where the focus is on the analysis of design efficiency, our goal in this article is to introduce an integrated algorithm-architecture design framework that guarantees the correctness of the integrated design in the early stages of the lifecycle.

A hierarchical modelling framework for communication architectures of multiprocessing SoCs is proposed in [22]. A hierarchical class library is used to develop communication architectures with incremental effort by adapting object-oriented analysis and design methods. The hierarchical approach of evaluating communication architectures in the proposed framework of [22] is similar to ours. However, it concentrates only on design exploration of network architectures, without considering network algorithms.

There is a large number of successful approaches that apply the refinement technique in producing a correct model of a system under development. Action Systems [8] - a design methodology based on refinement- is used in [19] to propose a formal framework for the design of asynchronous pipelined microprocessors. Another design methodology is proposed in [10] to integrate the Event-B formal method into the SoC design. The role of refinement in SoC development is demonstrated by specifying the system requirements and developing models of the system gradually, while the consistency of the various models is preserved. Using the refinement technique to produce a correct high-level design of a system in [10, 19] is similar to our design methodology.

The Event-B formal method has been widely employed for formal development and analysis of different network protocols. An algorithm for network topology discovery [11] is modelled and verified in Event-B. Stepwise and event-based approaches for modelling and analysing dynamic network protocols are proposed in [7, 6, 12]. A correct-by-construction development of a NoC model is proposed in [15, 14]. In this paper we employ the same formal method (i.e., Event-B) in order to develop system models. However, while in the above articles the focus is on modelling and analysing particular network algorithms, hence we introduce a framework for modelling network designs, with an approach integrating algorithm and architecture analysis.

## 3. Event-B

In this section we shortly introduce Event-B as well as an extension mechanism in Event-B that we employ in this paper. Event-B [3] is a formal method devised for modelling and reasoning about parallel, distributed and reactive systems. Event-B comes with tool support in the form of the Rodin

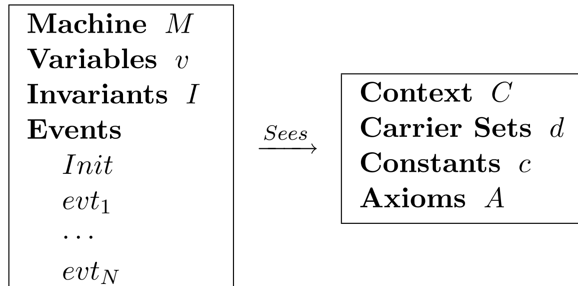



Figure 1: A machine $M$ and a context $C$ in Event-B

platform [2, 21, 20, 4], which provides automated support for modelling and verification by theorem proving.

### *3.1. Event-B Syntax and Semantics*

**Event-B Syntax..** In Event-B, the notion of a *machine* [20] operating on an *abstract state* is used to specify a system. The state models the problem domain of the system via a collection of variables and is modified by defined operations on this state. Thus, it describes the behaviour of the specified system, also refereed to as the dynamic part. A system model may also have an accompanying component, called *context*, which contains the static part of the system. Within a context, user-defined sets and constants are declared and their properties and relationships are defined in a list of model *axioms*. The general structure of an Event-B model is illustrated in Fig. 1. The relationship between a machine and its accompanying context is expressed by the keyword *Sees*, denoting a structuring technique that allows the machine access to the contents of the context.

A machine with the unique name $M$ has a list of state variables $v$, declared in the **Variables** clause and initialized by the *Init* event. The variable types are declared by the constraining predicates $I$ given in the **Invariants** clause. Moreover, the invariant clause may contain other predicates defining properties that should be preserved over the state of the model. The operations on the variables are given as a set of atomic events specified in the **Events** clause. Generally, an event is defined in the following form:

$$evt \;\widehat{=}\; \textbf{any}\ vl\ \textbf{where}\ g\ \textbf{then}\ S\ \textbf{end},$$

where the variable list $vl$ contains local variables visible within the event and

the guard $g$ represents a state predicate over the state variables $v$ and $vl$. The action $S$ is a statement describing how the event affects the system state and is given in the form of deterministic or non-deterministic assignment over the system variable. A deterministic assignment, $x := E(x, y)$, has the standard syntax and semantics. A non-deterministic assignment is denoted either as $x :\in Set$, where $Set$ is a set of values, or as $x :| \ P(x, y, x')$, where $P$ is a predicate relating initial values of $x, y$ to some final value $x'$. As a result of a non-deterministic assignment, $x$ can get any value belonging to $Set$ or according to $P$.

The occurrence of events represents the observable behaviour of the system. The event guard defines the conditions under which the action can be executed, i.e., when the event is *enabled*. If several events are enabled at the same time, any of them can be chosen for execution non-deterministically. If none of the events is enabled then the system deadlocks.

The modelling of a large system in Event-B is based on $refinement$, i.e., verified modelling in a stepwise manner. Modelling starts from an abstract system specification that models some of the essential functional requirements. While capturing more detailed requirements, the model becomes richer by developing the abstract states and state changes, i.e., introducing new variables and events. This type of model refinement is called *superposition* refinement. The superposition refinement steps are complete when all the requirements of the system have been taken into account in the model. Another kind of refinement is called $data\ refinement$ and in this case abstract variables and events can be replaced with their more concrete counterparts. In a data refinement step, we can remove some variables and add new ones. Therefore, the invariant of a refined model should define the relationship between the abstract and concrete variables; this type of invariants are called *gluing invariants*. In other words, the state of an abstract machine is related to the state of a concrete machine by a gluing invariant. In this article we employ both the superposition and data refinement.

The event structure in a refined machine may contain the abstract event clauses and two other clauses as shown in the following form:

$$ref_evt \ \widehat{=} \ \textbf{refines}\ abs_evt\ \textbf{any}\ vl\ \textbf{where}\ g\ \textbf{with}\ w\ \ \textbf{then}\ S\ \textbf{end},$$

where the list of the abstract events, *abs_evt*, that this event refines are enumerated in the **refines** clause. The **with** clause contains the *witness* of the

corresponding abstract event. The witness $w$ is declared in a refining event for each disappearing parameter of the abstract event. The witness for parameter $a$ is a predicate $P(a)$. A witness predicate can be either deterministic or non-deterministic. A deterministic witness $P(a)$ is of the form $a = E$.

**Event-B Semantics..** Each construct in Event-B has a well-defined semantics. Additionally, machines must satisfy a set of constraints so that ensure that the specification is internally consistent and also ensure that the behaviour of a refined machine is consistent with the behaviour of the machines it refines.

There are three principal constructions, called *substitutions*, to change the state of a machine. The semantics of substitution in Event-B is defined using before-after (BA) predicates. Before-after predicates contain primed and unprimed variables where the primed variables represent the after value of a variable and the unprimed variables the before value. In general, the substitution into a predicate for each of the three constructions takes the form illustrated in Fig. 2

| Substitution | [Substitution]R |
|---|---|
| $v := E$ | $[v := E]R$ |
| $v :\mid P$ | $\forall v' \cdot P \Rightarrow [v := v']R$ |
| $v :\in S$ | $\forall v' \cdot v' \in S \Rightarrow [v := v']R$ |

Figure 2: Substitution

where $v$ is a list of variables and $E$ a list of expressions; $P$ is a predicate containing both $v$ and $v'$; $v'$ represents the value of $v$ after the action.

In an Event-B event, guard is a predicate that identifies a set of pre-state and action identifies a set of post-states. A before-after predicate relates these pre-states and post-states for each event as follows:

$$BA(evt) \quad = \quad \exists vl.\ g \ \wedge \ BA(S).$$

The semantics of a whole Event-B model is formulated as a number of *proof obligations*, expressed in the form of logical sequent. Below we describe only the most important proof obligations that should be verified (proved) for the initial and refined models. The full list of proof obligations can be found in [3].

Every Event-B model should satisfy the event feasibility and invariant preservation properties. For each event of the model, $evt_i$, its feasibility means that, whenever the event is enabled, its before-after predicate (BA) is well-defined, i.e., there exists some reachable after-state:

$$A(d,c),\ I(d,c,v),\ g_i(d,c,v)\ \vdash\ \exists v' \cdot BA_i(d,c,v,v') \tag{FIS}$$

where $A$ stands for the conjunction of the model axioms, $I$ is the conjunction of the model invariants, $g_i$ is the event guard, $d$ stands for the model sets, $c$ are the model constants, and $v, v'$ are the variable values before and after event execution.

Each event $evt_i$ of the Event-B model should also preserve the given model invariant:

$$A(d,c),\ I(d,c,v),\ g_i(d,c,v),\ BA_i(d,c,v,v')\ \vdash\ I(d,c,v') \tag{INV}$$

The invariant preservation proof obligation verifies that each concrete invariant is preserved by each pair of concrete and abstract events. Since the initialisation event has no initial state and guard, its proof obligation is simpler:

$$A(d,c),\ BA_{Init}(d,c,v')\ \vdash\ I(d,c,v') \tag{INIT}$$

The formal semantics provides us with a foundation for establishing correctness of Event-B specifications. In particular, to verify correctness of a specification, we need to prove that its initialisation and all events preserve the given invariant.

Besides, to verify correctness of a refinement step, we need to prove a number of proof obligations for a refined model. We need to verify that the behaviour of a refined machine is consistent with the behaviour of the machine it refines. For brevity, here we show only a few essential ones.

Let us first introduce a shorthand $H(d,c,v,w)$ to stand for the hypotheses $A(d,c),\ I(d,c,v),\ I'(d,c,v,w)$, where $I,\ I'$ are respectively the abstract and refined invariants, and $v,\ w$ are respectively the abstract and concrete variables. Then the feasibility refinement property for an event $evt_i$ of a refined model can be presented as follows:

$$H(d,c,v,w),\ g'_i(d,c,w)\ \vdash\ \exists w'.\ BA'_i(d,c,w,w') \tag{REF_FIS}$$

where $g'_i$ is the refined guard and $BA'_i$ is a before-after predicate of the refined event.

The event guards in a refined model can be only strengthened in a refinement step:

$$H(d, c, v, w),\ g'_i(d, c, w)\ \vdash\ g_i(d, c, v) \qquad \text{(REF_GRD)}$$

where $g_i, g'_i$ are respectively the abstract and concrete guards of the event $evt_i$. The purpose of the guard strengthening proof obligation is to ensure that the concrete guards in the refining event are stronger than the abstract ones.

Finally, the *simulation* proof obligation (REF_SIM) requires to show that the "execution" of a refined event is not contradictory with its abstract version:

$$H(d, c, v, w),\ g'_i(d, c, w),\ BA'_i(d, c, w, w') \vdash \exists v'. BA_i(d, c, v, v') \wedge I'(d, c, v', w') \qquad \text{(REF_SIM)}$$

where $BA_i, BA'_i$ are respectively the abstract and concrete before-after predicates of the same event $evt_i$. The simulation proof obligation ensures that each action in a concrete event simulates the corresponding abstract action.

The model verification effort and, in particular, automatic generation and proving of the required proof obligations, are significantly facilitated by the provided tool support – the Rodin platform [1, 21, 20, 4].

Let us note here the quintessential feature of Event-B and its associated Rodin platform. Modelling in Event-B is semantically justified by proof obligations. Every update of a model generates a new set of proof obligations in the background. It is this interplay between modelling and proving that sets Event-B apart from other formalisms. Without proving the required obligations, we cannot be sure of correctness of a model. The proving effort thus encourages the developer to structure formal model development in such a way that manageable proof obligations are generated at each step. This leads to very abstract initial models so that we can gradually introduce into a system model various facets of the system. Such a development method fits well when we have to describe complex algorithms.

The interplay between modelling and proving characteristic of Event-B provides suitable formal means to reason about and finding out the design problems. The perspective of Event-B to modelling matches to our purpose of developing a formal framework for network designs because it allows to construct a correct framework, in each abstract level, by modelling and proving correctness of the model in a cyclic way, as shown in Fig. 3.

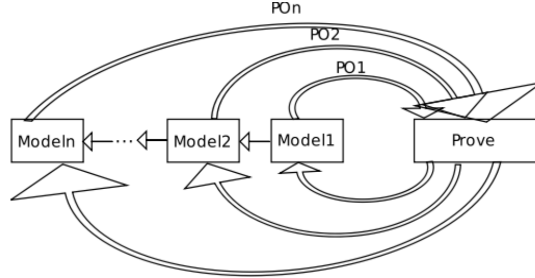



Figure 3: Formal Development in Event-B

*3.2. Event-B Mathematical Extension*

**Extension Syntax..** We first outline the syntax of the proposed extension mechanism [5, 9] in Event-B, already implemented in Rodin. In particular we emphasize the *theory* component, based on a rule-based prover plug-in; this component allows the declaration of user-defined new predicates, new operators, new inductive data types and new rewrite rules.

A theory component has a name, a list of global type parameters, and an arbitrary number of definitions and rules for modelling predicates, operators, and inductive data types. The general structure of a theory component is shown in Fig. 4. In the following, we briefly overview the general structure of defining predicates and operators.

**theory** *name*
**type parameters** $T_1, ..., T_n$
{ $\langle$*Predicate Definition*$\rangle$
| $\langle$*Operator Definition*$\rangle$
| $\langle$*DataType Definition*$\rangle$
| $\langle$*Rewrite Rule*$\rangle$
| $\langle$*Inference Rule*$\rangle$ }

Figure 4: Theory Component

A predicate has a name and can be declared as either *infix* or *prefix* with a number of arguments. The *condition* clause specifies a well-definedness condition and the argument types are also inferred from the condition. The definition of the new predicate is provided by the *definition* clause. The structure of a predicate definition is shown in Fig. 5.

An operator returns an expression based on a number of expressions. It can be declared as either *infix* or *prefix* with a number of arguments.

**predicate** *Identifier*
  **(prefix** or **infix)**
  **args** $x_1, ..., x_n$
  **condition** $P(x_1, ..., x_n)$
  **definition** $Q(x_1, ..., x_n)$

Figure 5: Predicate Definition

The *condition* clause specifies a well-definedness condition and the argument types are also inferred from the condition. The *definition* clause defines the expression $E(x_1, ..., x_n)$. The structure of an operator definition is shown in Fig. 6.

**operator** *Identifier*
  **(prefix** or **infix)**
  **args** $x_1, ..., x_n$
  **condition** $P(x_1, ..., x_n)$
  **definition** $E(x_1, ..., x_n)$

Figure 6: Operator Definition

**Extension Semantics..** The mathematical extension mechanism of Event-B supports three types of user-defined extensions of the mathematical language and theory. One type is set-theoretic expressions or predicates extensions, the second is the rule library for predicates and operators extensions, and the last is extensions of the set theory itself through the definition of algebraic types by using new set constructors. Out of these three extensions, we discuss below the rule library for predicates and expressions which are used in this article to introduce the theory of our framework.

In Event-B, predicates and expressions are introduced as distinct syntactic categories. Predicates are defined in terms of basic predicates ($\top$, $\bot$, $A = B$, etc), predicate combinators ($\neg$, $\wedge$, $\vee$, etc) and quantifiers ($\forall$, $\exists$). Expressions are defined in terms of constants ($0$, $\varnothing$, etc), variables ($x$, $y$, etc) and operators ($+$, $\cup$, etc). To ensure that mathematical formulas made of predicates and expressions are meaningful, they come with type checking. Each expression in a formula is associated with a type which denotes the set of values that the expression can take and it can be in one of the following forms:

- a basic set, that is a predefined set (Z or BOOL) or a carrier set provided by the user;
- a power set of another type $\alpha$, $\mathbb{P}(\alpha)$;
- a cartesian product of two types $\alpha$ and $\beta$, $\alpha \times \beta$

Upon associating a type to each expression the formula is checked by some type checking rules to prove that it is well-typed. Checking rules enforce that the operators used can be meaningful. As the type checking is a static check, we cannot always prove that a formula is meaningful. Therefore, for some operators, some additional dynamic constraints (specified in the well-definedness dynamic checks) are verified. Detecting a meaningless formula is done by disproving some well-definedness lemma. Well-definedness (WD) lemmas are produced by a $WD$ operator that takes a formula as argument. The complete list of the $WD$ operators for basic predicates and expressions of the Event-B kernel can be found in [18].

In the following, we outline how new predicates and expressions are specified and how their corresponding rules are formed. The corresponding rules are typing rules and well-definedness rules, as discussed above. A predicate *pred* is defined in terms of existing predicates and rules for typing the predicate arguments $x_1, \ldots, x_n$ as shown in the following table:

| Predicate | Type rule | Definition |
|---|---|---|
| $pred(x_1, \ldots, x_n)$ | $\mathbf{type}(x_1) = \alpha_1 \quad \ldots \quad \mathbf{type}(x_n) = \alpha_n$ | $P(x_1, \ldots, x_n)$ |

In predicate definition $P(x_1, \ldots, x_n)$, we cannot refer to the newly introduced predicate *pred*.

A new operator is defined in terms of an expression along with typing and well-definedness rules. An operator has a name *op* and a list of expression arguments $x_1, \ldots, x_n$ as shown in following table:

| Expression Operator | Definition |
|---|---|
| $op(x_1, \ldots, x_n)$ | $op(x_1, \ldots, x_n) = E(x_1, \ldots, x_n)$ |

In the operator definition, the expression $E$ should not refer to the newly defined operator *op*.

The typing rule for the operator *op* is used by a type checker for expressions involving *op*. The type $\alpha$ for a new operator *op* is introduced as follows:

| Expression Operator | Type rule |
|---|---|
| $op(x_1, \ldots, x_n)$ | $\frac{\mathbf{type}(x_1)=\alpha_1 \quad \ldots \quad \mathbf{type}(x_n)=\alpha_n}{\mathbf{type}(op(x_1,\ldots,x_n))=\alpha}$ |

An operator well-definedness $WD(op)$ depends on the well-definedness of its arguments and possible additional conditions $P(x_1, \ldots, x_n)$. However, some operators have no additional condition and their well-definedness depends on the well-definedness of their arguments. The well-definedness rule for a new operator is as follows:

| Expression Operator | Well-definedness |
|---|---|
| $\mathbf{WD}(op(x_1, \ldots, x_n))$ | $\mathbf{WD}(x_1), \ldots, \mathbf{WD}(x_n), P(x_1, \ldots, x_n)$ |

New predicates and operators extend the core mathematical language to facilitate specification and verification of systems. Moreover, having the new introduced predicates and operators, we can specify and prove theorems that can be used in verifying properties of systems. In this article, we use the Event-B extension mechanism to introduce a basic theory of networks. The foundation of the network designs is extracted from a hierarchical framework, introduced in the following sections.

## 4. The Structure of the Hierarchical Framework NetCorre

In this section we introduce our approach to integrating the algorithm development and architecture design. Our central interest is in devising correct network designs and so we put forward the verification of each of these two components of modelling as well as their integration. Therefore, a formal methodology model of network design, proposed as a framework is discussed in this chapter. The skeleton of the framework is in fact composed of three basic components: `Network Architecture`, `Network Primitives`, and `Network Properties`. This is illustrated in Fig. 7.

The `Network Architecture` component describes the network elements, their relationship to each other, and various other constructs needed for communication. It consists of two sub-components as illustrated in Fig. 7: a `Structural Elements` sub-component and a `Relationships` sub-component. Further on, we identify three main types of structural elements involved in network designs, namely `Node`, `Channel` and `Buffer`. A node denotes any type of communicating unit. A channel connects nodes together. A buffer stores data temporarily in the start and end points of the channels. Different functionality can be mapped to each of these structural elements but any of them shares certain common characteristics, commonly inherited from the higher

abstract elements. In this article, the higher abstract structural elements proposed in the framework can be reused to develop more concrete elements. In other words, the common characteristics are introduced and the differences are encapsulated as block boxes in structural elements. The relationships between the structural elements in the network shows the topology of the architecture.

The `Network architecture` shows how different elements are connected to each other in order to transfer messages. The message transfer methods are described in the `Network Primitives` component of our framework. We identify four main network primitives, namely `Injecting`, `Switching`, `Routing` and `Receiving`, as illustrated in Fig. 7. Injecting refers to introducing a new message in the network. Routing refers to deciding the route that a message should follow. Switching determines when the routing decisions are made, the setting/resetting of the switches inside the nodes, and how the messages are transferred inside the nodes. Receiving is the function that deletes the messages having reached their destination from the network. The `Network Primitives` subcomponents proposed in the framework share certain common features for algorithm development. In other words, these are reusable modules that are inherited and refined in the development of a network algorithm.

The `Network Properties` component is intended for the modelling and verification of the network design properties, based on the given architecture and primitives. The aim of this component is to demonstrate that all components of the framework meet their specifications. Moreover, the interaction and integration satisfies the general correctness properties. A typical property for the functional correctness of communication is that all the injected messages are received by all their destinations. However, other properties can be modelled as well, for instance non-functional properties such as the network performance or energy consumption of a given network design. In this article we only refer to the functional correctness of communication. To verify the general correctness of the integration in the framework we must analyse the possible interactions of the elements of the `Network Architecture` and the `Network Primitives` to see whether the combined design satisfies the specification.

Hence, in the proposed framework we concentrate on developing the fundamental components of a network which are common between all network designs. Therefore, we verify the interactions of these components and we construct the framework in such a a way that the correctness of the frame-

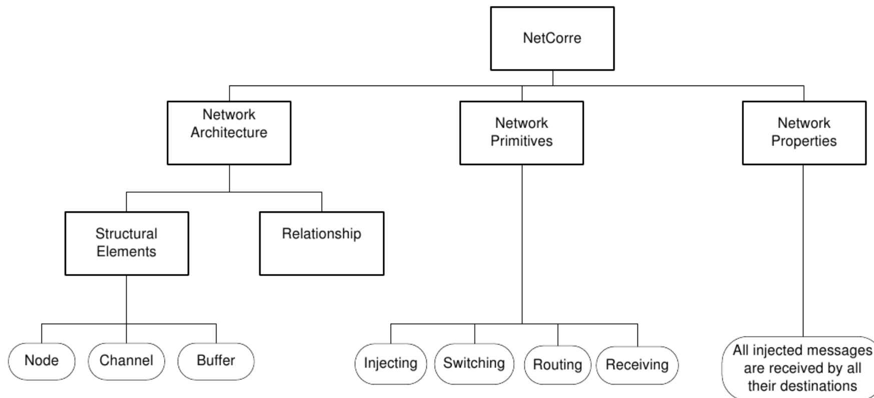



Figure 7: Network Scheme

work is guaranteed. The components of the framework can be refined by designers to construct different network models. However, the designer does not need to prove all the properties from the beginning; all the proved properties in the framework can be reused and only the newly added states must be verified. We note that using pre-proved components of the framework alleviates the difficulties of verification and also decreas the design lifecycle. In this paper, we additionally provide a library of rules for using the pre-proven fundamental components in order to develop network designs.

## 5. The stepwise construction of NetCorre in Event-B

In this section, we describe the modelling of the proposed structure of network designs in Event-B and discuss its correctness. The network architecture describes the static part of a network, so all its elements and properties are specified in the context part of the model as sets, constants, and axioms. The network primitives define the dynamic behaviour of the communication network, therefore are modelled in the machine part of the Event-B model as variables and events. The network properties are modelled as a list of invariants in the machine part of the model and verified via proof obligations. In the following we outline this development.

We model the network design elements in a stepwise manner, employing the abstraction and refinement concepts put forward in Section 3. For every abstraction level we have a corresponding context, machine, list of invariants, and proof obligations to discharge. The initial model consists of the

least number of structural elements so that the correctness property can be formulated. In the subsequent levels, we gradually add the other structural elements so that they refine the concepts in the previous models. As we hierarchically refine the architecture, the network primitives component is also hierarchically refined. In each abstraction level we verify the network properties, expressed in terms of the (hierarchically-)corresponding architectural elements and primitives. The stepwise development continues until all the elements of the network design are modelled. The resulting model can then be employed to specify other structural elements as well as specific network designs.

### *5.1. Specification of the NetCorre: Level 1*

At Level 1 of abstraction, we consider just a small number of structural and primitive elements, essentially as few as possible for expressing relevant invariants. This means that some elements are abstracted away from this initial model. More precisely, in Level 1 we model the structural element `Node` by defining a finite (**@axm1**) and non-empty (**@axm2**) set *NODE* that models the network elements. The relationships between these nodes, defining the network topology, are modelled with a constant named *Neigh*. This constant effectively records the pairs of related nodes as shown by **@axm3** in Fig. 8. The relation *Neigh* denotes a set of non-empty (**@axm4**) and bidirectional node links that are symmetric (**@axm5**) and irreflexive (**@axm6**). In order to model the functional correctness of the network, we need to specify that all the injected messages are received by all their destinations. For this, we model another finite (**@axm7**) and non-empty (**@axm8**) set, $MESSAGE$. A message in the network has a source node and a set of destination nodes, hence any element of $MESSAGE$ is conceptually a triple $(msgID, src, \{des\})$. Here we model this conceptual triple via the $MESSAGE$ set together with two functios for the source and destinations of messages. The functions are defined by **@axm9-10**. To specify a generic model, we consider having several destinations for any message. It lets us model both uni-casting and multi-casting in a network.

We model the corresponding network primitives in Level 1 with variables and events as illustrated in Fig. 9. Thus, in the initial model we have four abstract variables: $snt_pl, mv_pl, rcv_nd$ and $rcv_pl$. These variables are sufficient for modelling and verifying the functional correctness of the network. The *snt_pl* variable (abbreviation for 'sent pool') denotes the list of messages which are injected by a source node to the network (**@inv1**). The

**SET** $NODE \quad MESSAGE$
**CONSTANTS** $Neigh \quad src \quad des$
**AXIOMS**
**@axm1** $finite(NODE)$
**@axm2** $NODE \neq \varnothing$
**@axm3** $Neigh \in NODE \leftrightarrow NODE$
**@axm4** $Neigh \neq \varnothing$
**@axm5** $Neigh = Neigh^{-1}$
**@axm6** $dom(Neigh) \lhd id \cap Neigh = \varnothing$
**@axm7** $finite(MESSAGE)$
**@axm8** $MESSAGE \neq \varnothing$
**@axm9** $src \in MESSAGE \rightarrow NODE$
**@axm10** $des \in MESSAGE \rightarrow \mathbb{P}(NODE)$

Figure 8: The Constext Part (Level 1): ctx1

*mv_pl* variable (abbreviation for 'moving pool') denotes the current position of messages still travelling in the network (**@inv2**). The *rcv_nd* variable (abbreviation for 'receive node') models the instances of messages having reached destinations (**@inv6-7**). When a message is received to one of its destinations, a corresponding map is added to the *rcv_nd* relation. The *rcv_pl* variable (abbreviation for 'receive pool') denotes the list of messages which are received by all their destinations and consequently removed from the network (**@inv3**); this means that a message-to-destination map for all the destinations of that message should be in *rcv_nd*.

The four network primitives are abstractly modelled in Level 1 by three events: *inject_msg*, *travel_msg*, and *receive_msg*. The *inject_msg* event handles the injecting primitive by adding a new message (**@grd1-2**) to the *snt_pl* set (**@act1**) and the *mv_pl* set (**@act2**). The reason for adding the message to *mv_pl* is to launch the message transferring process.

The *travel_msg* event handles both the switching and the routing primitives in Level 1, by modelling the transferring of a message from a node to other nodes. As the framework covers both unicast and multicast communication, we need to model the transferring of a message from a node to several nodes. Hence, in the *travel_msg* event, we define a local variable *r_n* that models an arbitrary subset of the neighbours of a node (**@grd2**). As we define a subset of neighbours (**@grd2**), it could be possible that *r_n* is emptyset which then causes the message to disapear from the network. In order to avoid this, we add a new guard (**@grd3**) to restrict the *r_n* variable. Namely, when a message is not received by its destination, it should be

**invariants**
  **@inv1** $snt_pl \subseteq MESSAGE$
  **@inv2** $mv_pl \in snt_pl \leftrightarrow NODE$
  **@inv3** $rcv_pl \subseteq MESSAGE$
  **@inv4** $rcv_pl \cap dom(mv_pl) = \varnothing$
  **@inv5** $dom(mv_pl) \cup rcv_pl = snt_pl$
  **@inv6** $rcv_nd \in NODE \leftrightarrow MESSAGE$
  **@inv7** $\forall msg, d \cdot d \mapsto msg \in rcv_nd \Rightarrow d \in des(msg)$
  **@inv8** $\forall msg \cdot msg \in dom(mv_pl) \Rightarrow (\exists d \cdot d \in des(msg) \land d \mapsto msg \notin rcv_nd)$
  **@inv9** $\forall msg \cdot msg \in rcv_pl \Leftrightarrow (\forall d \cdot d \in des(msg) \Rightarrow d \mapsto msg \in rcv_nd)$
  **@inv10** $ran(rcv_nd) \subseteq snt_pl$
  **@inv11** $\forall msg \cdot msg \in snt_pl \land des(msg) \setminus rcv_nd^{-1}[\{msg\}] \neq \varnothing \Rightarrow msg \in dom(mv_pl)$
  **@inv12** $mv_pl = \varnothing \Rightarrow ran(rcv_nd) = snt_pl$

**inject_msg**
**any** $msg$
**where**
  **@grd1** $msg \in MESSAGE$
  **@grd2** $msg \notin snt_pl$
**then**
  **@act1** $snt_pl := snt_pl \cup \{msg\}$
  **@act2** $mv_pl := mv_pl \cup \{msg \mapsto src(msg)\}$
**end**

**travel_msg**
**any** $msg$ $n$ $r_n$
**where**
  **@grd1** $msg \mapsto n \in mv_pl$
  **@grd2** $r_n \subseteq Neigh[\{n\}]$
  **@grd3** $(n \notin des(msg) \land r_n \neq \varnothing) \lor (n \mapsto msg \in rcv_nd \land (\exists m \cdot msg \mapsto m \in mv_pl \land n \neq m)) \lor (n \mapsto msg \in rcv_nd \land (\forall m \cdot msg \mapsto m \in mv_pl \Rightarrow n = m) \land r_n \neq \varnothing)$
**then**
  **@act1** $mv_pl := (mv_pl \setminus \{msg \mapsto n\}) \cup (\{msg\} \times r_n)$
**end**

**received_msg**
**any** $msg$ $n$ $rcv$
**where**
  **@grd1** $msg \mapsto n \in mv_pl$
  **@grd2** $n \in des(msg)$
  **@grd3** $rcv \subseteq MESSAGE$
  **@grd4** $((\exists d \cdot d \in des(msg) \land d \mapsto msg \notin rcv_nd \cup \{n \mapsto msg\}) \land rcv = \varnothing) \lor ((\forall d \cdot d \in des(msg) \Rightarrow d \mapsto msg \in rcv_nd \cup \{n \mapsto msg\}) \land rcv = \{msg\})$
**then**
  **@act1** $rcv_nd := rcv_nd \cup \{n \mapsto msg\}$
  **@act2** $rcv_pl := rcv_pl \cup rcv$
  **@act3** $mv_pl := rcv \lhd mv_pl$
**end**

Figure 9: The Machine Part (Level 1): M1

routed next and should not be removed from the network. This condition is modelled by the first *or* (∨) part of the **@grd3**. The second *or* part of the **@grd3** models the state when the current position of the message is one of its destinations and there are other instances of the message in the network. In this case $r_n$ can be either an emptyset or a subset of neighbours. The last *or* part of the **@grd3** restricts the $r_n$ to a non-emptyset when the current position is one of its destinations and the last instance of the message is in this position. Without the last condition of **@grd3**, we could remove the message from the network when the message is not yet received by all its destinations. We note that we postpone the modelling of specific routing and switching primitives for the subsequent refinement steps; however, we add the necessary conditions so that any switching and routing primitives can be derived out of the *travel_msg* event.

The *receive_msg* event handles the receiving primitive of the network model in two different situations. Either the message is received by all its destinations or, there are still some destinations that did not receive the message. To specify two different situations in one event, we define a local variable *rcv*, which is a subset of messages (**@grd3**). When there are still some destinations that did not received the message, the *rcv* variable is set to empty, as shown by the first *or* part of **@grd4**. When the message is received by all its destinations, the *rcv* variable is set to the message, shown by the second *or* part of **@grd4**. When there are still some destinations that did not received the message, the corresponding message-to-destination map is added to *rcv_nd* (**@act1**) and the message is not added to *rcv_pl* because there are still destinations waiting for it. When the message is received by all its destinations, the message is also added to the *rcv_pl* (**@act2**) and removed from the *mv_pl* (**@act2**).

To guarantee the functional correctness of our modelling, we specify the properties of the network as a list of invariants shown in Fig. 9. The first three invariants (**@inv1-3**) and **@inv6** are for variable definitions and the remaining invariants illustrate the correctness properties of network that should be preserved by all the events. While messages are travelling in the network, they are not received by all their destinations (**@inv8, 11**) and when a message is received at all its destinations, then there should not be any instance of that message in the network, shown by **@inv4**; moreover, it should be in the received state, shown by **@inv9**. As we model a network without message loss, we should guarantee that an injected message is either in the travelling state or received state and this property is shown by **@inv5**. The *rcv_nd*

models messages which are received at destination: the $rcv_nd$ relation can only consist of $node - msg$ pairs where $node$ is one of the destinations of $msg$ (shown by **@inv7**). When no messages is in the travelling state it means that all the injected messages are received by all their destinations, as formulated by **@inv12**.

The correctness verification of the initial model in which only the `Node` subcomponent of the `Structural Elements` is specified is performed by proving the satisfiability of invariants by events and the well-definedness of the events' guards. The number of proof obligations that are generated and the number of proof obligations that either automatically or manually discharged are shown in Table. 1.

Table 1: Proof Statistics Level 1

| Model | Number of Proof Obligations | Automatically Discharged | Interactively Discharged |
|---|---|---|---|
| $Level1$ Context | 2 | 2(100%) | 0(0%) |
| $Level1$ Machine | 39 | 32(82%) | 7(18%) |

The initial model forms the first hierarchical level, Level 1, of our development. Level 1 is a foundation model for the subsequent refinement steps, where we refine the context part of the model to extend the network architecture and refine the machine part of the model to refine the network primitives. The relation between our refinement levels is illustrated in Fig. 10.

*5.2. Specification of the NetCorre: Level 2*

In Level 2, we extend the initial context to model the `Channel` structural element. We specify the set $CHANNEL$ that (physically) connects neighbour nodes to each other. In other words, for any pair in the $Neigh$ relation, we model a channel that connects the pair by modelling two functions: $start_ch_nd$ (abbreviation for 'start channel from node') (**@axm11**) and $end_ch_nd$ (abbreviation for 'end channel to node') (**@axm12**). In order to show the new channel relations correspond to the neighbour relations, we specify that the bidirectional links between two neighbours corresponded to these new relations (**@axm13**). Moreover, we model that the composition of these two relations is equal to $Neigh$ relation (**@axm14**) as shown in Fig. 11.

At this level we refine the network primitive component to specify the transferring of messages through the channels. In this level of the hierarchy, we use the data refinement technique. In the initial model, the $mv_pl$ relation denotes the current position of messages in the network. In order to

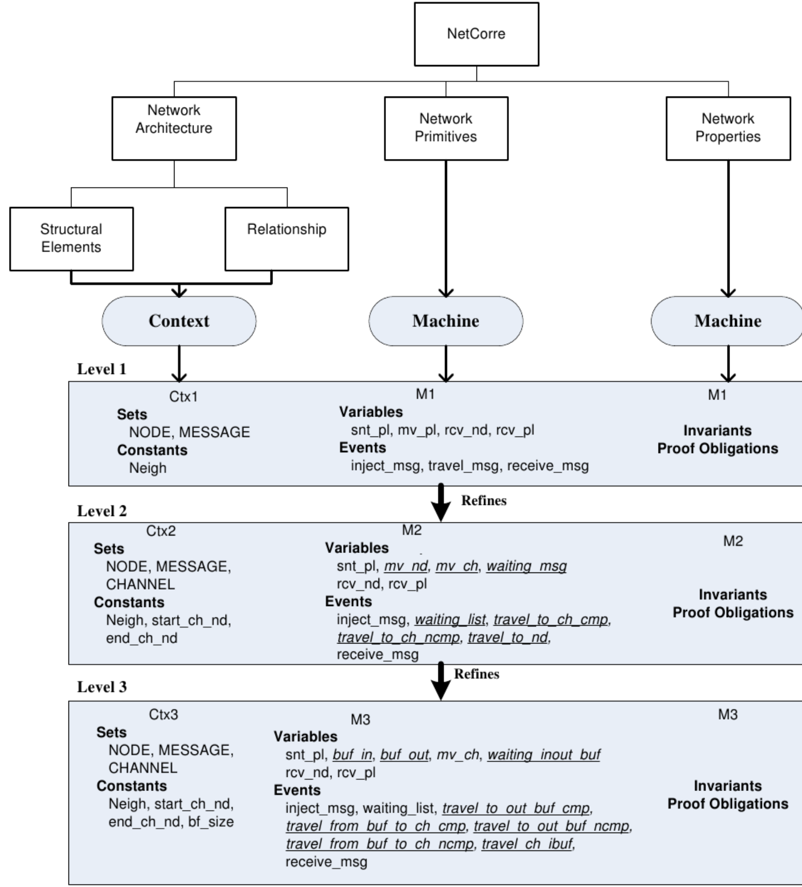



Figure 10: Network Scheme in Event-B

**SETS** $CHANNEL$
**CONSTANTS** $start_ch_nd$ $end_ch_nd$
**AXIOMS**
**@axm11** $start_ch_nd \in CHANNEL \rightarrow NODE$
**@axm12** $end_ch_nd \in CHANNEL \rightarrow NODE$
**@axm13** $\forall n, ch \cdot ch \mapsto n \in start_ch_nd \Rightarrow (\exists ch2 \cdot ch2 \mapsto end_ch_nd(ch) \in start_ch_nd$
$\land ch2 \mapsto n \in end_ch_nd)$
**@axm14** $Neigh \cup (dom(Neigh) \lhd id) = start_ch_nd^{-1}; end_ch_nd$

Figure 11: The Context Part (Level 2): ctx2

**invariants**
**@inv13** $mv_nd \in snt_pl \leftrightarrow NODE$
**@inv14** $mv_ch \in snt_pl \leftrightarrow CHANNEL$
**@inv15** $mv_ch^{-1} \in CHANNEL \nrightarrow snt_pl$
**@inv16** $(mv_ch; start_ch_nd) \cup mv_nd = mv_pl$
**@inv17** $\forall ch, msg \cdot msg \mapsto ch \in mv_ch \Rightarrow ((start_ch_nd(ch) \notin des(msg))$
$\vee (start_ch_nd(ch) \mapsto msg \in rcv_nd \wedge msg \notin rcv_pl))$

Figure 12: The Machine Part (Level 2): Invariants

represent the existence of messages in both nodes and channels, in the second model we replace the *mv_pl* variable with two variables: *mv_nd* (abbreviating for 'moving node pool') and *mv_ch* (abbreviating for 'moving channel pool'), shown in Fig. 12. The *mv_nd* variable denotes the current position of those messages that are in nodes ( **@inv13**) and the *mv_ch* variable denotes the current position of those messages that are in channels (**@inv14**). At any moment, any channel can be either free or busy with only one message, described by invariant (**@inv15**). Moreover, the union of messages in the $(mv_ch; start_ch_nd)$ and *mv_nd* relations is equal to those messages in *mv_pl*, described by the gluing invariant **@inv16** in Fig. 12. A message is transferred from a node to a channel if the node is not one of the message destinations or the node is a message destination but the message is not received by all its destinations, formulated by **@inv17**.

As a consequence of this data refinement, the events that model the network primitive component are updated, so that all the connections to *mv_pl* are substituted either by *mv_nd* or by *mv_ch*. In addition, we refine the *travel_msg* event by splitting it into two events: *travel_to_ch* and *travel_to_nd*, shown in Fig. 13. The *travel_to_ch* event models the transferring of a message from a node to a channel at the beginning of the channel. When there is a message at a node (*travel_to_ch*: **@grd1**), a subsets of output channels of the node (**@grd2**, correponding to **@grd2** in *travel_msg* in Level 1) which are free (**@grd4**) are chosen for transferring. The channel selection is restricted by considering three different situations corresponding to **@grd3** in *travel_msg* in Level 1. First, when the current node is not a destination of the message the subset of channels cannot be empty because the message should travel through another node, as described by the first *or* part of **@grd3**. Second, when the current node is one of the message destinations and another instance of a message in other nodes exists in the

network, then in this state, any arbitrary set of the output channels can be selected, as described by the second *or* part of **@grd3**. Last, when the current node is one of the message destinations and no instance of the message exists in the network, then the subset of channels cannot be empty because the message might still have some destinations to reach. This is described by the last *or* part of **@grd3**. As it is described, **@grd3** of the *travel_to_ch* event is the refinement of the **@grd3** of the *travel_msg* event in Level 1. In the *with* part of the *travel_to_ch* event, the witness is specified for the abstract parameter *r_n* of the abstract event *travel_msg* and its feasibility is given by axiom **@axm14**. This axiom is 'gluing' the constant *Neigh* from Level 1 to constants *start_ch_nd* and *end_ch_nd* from Level 2.

When the *travel_to_ch* event is executed the instance of the message in the current node is either removed from the *mv_nd* or remains in *mv_nd*, depending on whether the message is routed to all the selected channels according to a *routing* algorithm. In a state when all routes are not free the transferring the message should wait in the current node until the desired channels become free. This assignment (shown by **@act1**) is non-deterministic so that the event covers all possible situations of data transferring in a node. Finally, the transferring of a message from a node to the selected subset of channels is modelled by **@act2**.

The *travel_to_nd* event models the transferring of a message from a channel to a node at the end of the channel. The *travel_to_nd* event is enabled when there is a message in a channel (**@grd1**). This event removes the message from the channel (**@act1**) and adds it to the node at the end of the channel (**@act2**). As the abstract parameters $n$ and *r_n* are removed in this refined event, we define witnesses for them based on the new introduced parameter *ch* to be sure about the correctness of the refinement. The $n$ parameter is equal to the node at the start point of the channel and *r_n*, which models neighbours of node $n$, is equal to the node at the end of the channel, described in the *with* part of the *travel_to_nd* event. The use of witnesses adds clarity to a refined model and determining the required witnesses is often straightforward. The witnesses are also used to generate simpler invariant preservation proof obligations by decomposing proof obligations which, otherwise, would contain the existentailly quantified formula in their hypothesis.

This data refinement comes with a list of proof obligations that express its correctness. These obligations guarantee that the second model preserves the network properties component and its statistics are shown in Table 2.

**travel_to_ch**
**refines travel**
**any** $msg \quad n \quad ch$
**where**
**@grd1** $msg \mapsto n \in mv_nd$
**@grd2** $ch \subseteq start_ch_nd^{-1}[\{n\}]$
**@grd3** $(n \notin des(msg) \land ch \neq \varnothing) \lor (n \mapsto msg \in rcv_nd$
$\land(\exists m \cdot msg \mapsto m \in (mv_ch; start_ch_nd) \cup mv_nd \land n \neq m))$
$\lor(n \mapsto msg \in rcv_nd \land (\forall m \cdot msg \mapsto m \in (mv_ch; start_ch_nd) \cup mv_nd \Rightarrow n = m)$
$\land \ ch \neq \varnothing)$
**@grd4** $ran(mv_ch) \cap ch = \varnothing$
**with** $r_n : r_n \subseteq (start_ch_nd^{-1}; end_ch_nd)[\{n\}]$
**then**
**@act1** $mv_nd \ :\mid mv_nd' = mv_nd \setminus \{msg \mapsto n\} \lor mv_nd' = mv_nd$
**@act2** $mv_ch := mv_ch \cup (\{msg\} \times ch)$
**end**

**travel_to_nd**
**refines travel**
**any** $msg \quad ch$
**where**
**@grd1** $msg \mapsto ch \in mv_ch$
**with**
$n : n = start_ch_nd(ch)$
$r_n : r_n = end_ch_nd(ch)$
**then**
**@act1** $mv_ch := mv_ch \setminus \{msg \mapsto ch\}$
**@act2** $mv_nd := mv_nd \cup \{msg \mapsto end_ch_nd(ch)\}$
**end**

Figure 13: The Machine Part (Level 2): Events

**An Extra Refinement Step..** To avoid adding too much complexity to the specification, **@act1** in *travel_to_ch* is modelled as a non-deterministic assignment in Level 2. A message in a node can be sent to a number of output channels depending on the routing decisions. If a message is transferred to all the selected channels, the message is removed from *mv_nd*. Otherwise, the *mv_nd* remains unchanged. Here we add a new variable *waiting_msg* which models the list of output channels of any node that waits for messages to be transferred, as shown by **@inv18** in Fig. 14.

When a node makes a decision about routing a message to different output channels, the node sends the message to all the selected channels provided that the selected channels are free; in this case the instance of the message is removed from the node. Otherwise, the message is sent to the free selected channels and waits for the other selected channels to become free. Once the message is sent to all the selected channels, the instance of the message is re-

Table 2: Proof Statistics Level 2

| Model | Number of Proof Obligations | Automatically Discharged | Interactively Discharged |
|---|---|---|---|
| $Level2$ Context | 1 | 1(100%) | 0(0%) |
| $Level2$ Machine | 67 | 47(70%) | 20(30%) |

**invariants**
**@inv18** $waiting_msg \in mv_nd \leftrightarrow CHANNEL$
**@inv19** $\forall msg, ch \cdot msg \mapsto ch \in mv_ch \Rightarrow msg \mapsto start_ch_nd(ch) \mapsto ch \notin waiting_msg$
**@inv20** $\forall msg, n, ch \cdot msg \mapsto n \mapsto ch \in waiting_msg \Rightarrow start_ch_nd(ch) = n$

Figure 14: The Extra Refinement Step Machine Part: Invariants

moved from the node. The $waiting_msg$ variable denotes the messages which are not transferred to the selected channels. Once a message is transferred to a selected channel, the message, which was waiting for that channel, is removed from the $waiting_msg$, illustrated by **@inv19**. In addition, we need to guarantee that when a message is about to be sent to output channels of a node, this means that the node is at the start point of these selected channels, shown by **@inv20**.

At this level, we split (via $refinement$) the $travel_to_ch$ event to three $travel_to_ch_ncmpl$, $travel_to_ch_cmp$ and $waiting_list$ events to remove the non-deterministic assignment in the abstract event as shown in Fig. 15. We use the group refinement technique in this level to indicate the required event ordering, as provided by the group refinement technique. In our group refinement, the $waiting_list$ event is the first event in the trace and the $travel_to\ _ch_cmp$ is the last event in the trace.

In the $waiting_list$ event, when there is a message at a node (**@grd1**) and the routing decision has not been taken for the message in the node (**@grd3**), an arbitrary number of output channels of the node are selected (**@grd2**). The **@grd4** in $waiting_list$ event expresses the same condition as the **@grd4** in the abstract $travel_to_ch$ event. The $waiting_msg$ is updated by adding the triple of message, node and the selected output channels, shown by **@act1**.

After the $waiting_list$ event, either $travel_to_ch_cmp$ or $travel_to_ch_ncmp$ are enabled depending on the availability of the selected output channels. As removing a pair of message and node from $mv_nd$ means transferring the

message to all the selected output channels, we distinguish between transferring of the message to a number of selected output channels (modelled by the *travel_to_ch_ncmp* event) and transferring of the message to all selected output channels (modelled by the *travel_to_ch_cmp* event).

When all the output channels of a node, where a message is waiting for transferring, are free (**@grd2** of *travel_to_ch_cmp* event), the message is removed from the node (**@act1**), switched to the corresponding output channels (**@act2**) and removed from the *waiting_msg* list of the node (**@act3**). The witness of the *travel_to_ch_cmp* expresses that the *ch* parameter is substituted by $waiting_msg[\{msg \mapsto n\}]$.

Finally, the *travel_to_ch_ncmp* event is enabled when at least one of the selected output channels of a node is busy (**@grd2** of *travel_to_ch_ncmp* event). In this state, the message is switched to the free channels (**@act1**) and only corresponding messages for those free channels are removed from the *waiting_msg* (**@act2**).

*5.3. Specification of the NetCorre: Level 3*

In the third model (Level 3) we add in our framework the concept of the `Buffer` structural element by modelling a constant *bf_size* and refine accordingly the dynamic part of the model. Intuitively, at the start and at the end points of any channel, we now model a buffer of size *bf_size* that stores the messages. Each node consists of a number of input and output buffers depending on its input and output channels. Any message in a node settles in either an input buffer of the node or an output buffer of the node. Therefore, the *mv_nd* variable is replaced by the *buf_in* and *buf_out* variables, via data refinement. The *buf_in* variable denotes the content of the input buffers of nodes (**@inv21**) with their capacities limited to the buffer size *bf_size* (**@inv22**). The *buf_out* variable denotes the content of the output buffers for nodes (**@inv23**) with their capacities similarly limited to the buffer size *bf_size* (**@inv24**), shown in Fig. 16. To complete this data refinement step, we establish the connection between abstract variables and the more concrete variables by the **@inv25** gluing invariant. This invariant guarantees that all the mappings in the *mv_nd* relation in the abstract model are in either *buf_in* or *buf_out*. When we prove that the invariant is preserved by all the events in the machine, we guarantee the network correctness.

We perform another data refinement at this level in order to relate the waiting messages in *waiting_msg* to a corresponding buffer.We refine the *waiting_msg* variable by newly introduced *waiting_inout_buf* variable. The

**waiting_list** $[first]$ **refines travel_to_ch**
**any** $n$ $ch$ $msg$
**where**
**@grd1** $msg \mapsto n \in mv_nd$
**@grd2** $ch \subseteq start_ch_nd^{-1}[\{n\}]$
**@grd3** $msg \mapsto n \notin dom(waiting_msg)$
**@grd4** $n \notin des(msg) \vee (n \mapsto msg \in rcv_nd \wedge (\exists m \cdot msg \mapsto m \in (mv_ch; start_ch_nd) \cup mv_nd$
$\wedge n \neq m)) \vee (n \mapsto msg \in rcv_nd \wedge (\forall m \cdot msg \mapsto m \in (mv_ch; start_ch_nd) \cup mv_nd \Rightarrow n = m))$
**then**
**@act1** $waiting_msg := waiting_msg \cup (\{msg\} \times \{n\} \times ch)$
**end**

**travel_to_ch_cmp** **refines travel_to_ch**
**any** $n$ $msg$
**where**
**@grd1** $msg \mapsto n \in mv_nd$
**@grd2** $\forall ch \cdot ch \in waiting_msg[\{msg \mapsto n\}] \Rightarrow ch \notin ran(mv_ch)$
**With**
$ch : ch = waiting_msg[\{msg \mapsto n\}]$
**then**
**@act1** $mv_nd := mv_nd \setminus \{msg \mapsto n\}$
**@act2** $mv_ch := mv_ch \cup (\{msg\} \times waiting_msg[\{msg \mapsto n\}])$
**@act3** $waiting_msg := \{msg \mapsto n\} \lhd waiting_msg$
**end**

**travel_to_ch_ncmp** **refines travel_to_ch**
**any** $n$ $msg$
**where**
**@grd1** $msg \mapsto n \in mv_nd$
**@grd2** $\exists ch \cdot ch \in waiting_msg[\{msg \mapsto n\}] \wedge ch \in ran(mv_ch)$
**With**
$ch : ch = waiting_msg[\{msg \mapsto n\}] \setminus ran(mv_ch)$
**then**
**@act1** $mv_ch := mv_ch \cup (\{msg\} \times (waiting_msg[\{msg \mapsto n\}] \setminus ran(mv_ch)))$
**@act2** $waiting_msg := waiting_msg \setminus (\{msg\} \times \{n\} \times (waiting_msg[\{msg \mapsto n\}] \setminus ran(mv_ch)))$
**end**

Figure 15: The Machine Part (The Extra Refinement Step): Events

**invariants**
**@inv21** $buf_in \in CHANNEL \rightarrow \mathbb{P}(MESSAGE)$
**@inv22** $buf_in = \{c \mapsto i | c \mapsto i \in buf_in \wedge card(i) \leq bf_size\}$
**@inv23** $buf_out \in CHANNEL \rightarrow \mathbb{P}(MESSAGE)$
**@inv24** $buf_out = \{c \mapsto i | c \mapsto i \in buf_in \wedge card(i) \leq bf_size\}$
**@inv25** $\forall msg, n \cdot msg \mapsto n \in mv_nd \Rightarrow (\exists ch \cdot (ch \in start_ch_nd^{-1}[\{n\}] \wedge msg \in buf_out(ch))$
$\vee (ch \in end_ch_nd^{-1}[\{n\}] \wedge msg \in buf_in(ch)))$

**@inv26** $waiting_inout_buf = \{msg \mapsto ch | ch \in CHANNEL \wedge msg \in buf_in(ch)\}$
**@inv27** $\forall msg, ch \cdot msg \mapsto ch \in waiting_inout_buf$
$\Leftrightarrow msg \mapsto starts_ch_nd(ch) \in waiting_msg$
**@inv28** $\forall msg, ch \cdot msg \mapsto ch \in waiting_inout_buf \Rightarrow msg \notin buf_out(ch)$
**@inv29** $\forall msg, ch \cdot msg \in buf_out(ch) \Rightarrow msg \mapsto ch \notin waiting_inout_buf$
**@inv30** $\forall msg, ch \cdot msg \mapsto ch \in waiting_inout_buf$
$\Rightarrow (\exists c \cdot c \in end_nd_ch^{-1}[\{start_ch_nd(ch)\}] \wedge msg \in buf_in(c))$

Figure 16: The Machine Part (Level 3): Invariants

*waiting_inout_buf* variable is defined by a message to channel relation (**@inv26**) that expresses the output channels of a node where a message should be transferred. To ensure the correct refinement of *waiting_msg* in the refined model, the **@inv27** gluing invariant is defined to formulate the relation between *waiting_msg* and *waiting_inout_buf*. This invariant guarantees that all the mappings in the *waiting_msg* relation are in *waiting_inout_buf*.

Once a message is transferred from input buffers to output buffers in a node, it is removed from *waiting_inout_buf*, i.e., messages which are in output buffers of nodes do not exist in the waiting list for transferring from input buffers to output buffers (**@inv28, 29**). Another important property of this refinement step is that all the messages which are waiting in a node to be transferred from input buffers to output buffers were received by one of the input buffers of the node (**@inv30**).

At this level of abstraction, we refine the *travel_to_nd* event in the model to the *travel_ch_ibuf* event as shown in Fig. 17. The *travel_ch_ibuf* event models the receipt of a message from a channel (**@grd1**), provided that the buffer at the end of the channel has space to add a new message (**@grd2**). This event releases the channel (**@act1**) and adds the message to the input buffer of the node corresponding to the end of the channel (**@act2**).

In the previous abstract model, we cannot see how messages are transferred inside a node, but only that messages exist in either a node or a channel. We have already modelled the transferring of a message from a node to multiple channels by *travel_to_ch_cmp* and *travel_to_ch_ncmp* events. How-

**travel_ch_ibuf**
**refines travel_to_nd**
**any** $ch$ $msg$
**where**
**@grd1** $msg \mapsto ch \in mv_ch$
**@grd2** $card(buf_in(ch)) < bf_size$
**then**
**@act1** $mv_ch := mv_ch \setminus \{msg \mapsto ch\}$
**@act2** $buf_in(ch) := buf_in(ch) \cup \{msg\}$
**end**

Figure 17: The Machine Part 1 (Level 3): Events

ever, in this refinement level, we detail on transferring messages inside the nodes. We consider moving messages from input buffers to output buffers and then from output buffers to channels. This view of the system gives the idea of splitting each of the $travel_to_ch_cmp$ and $travel_to_ch_ncmp$ events to two events. Therefore, we refine the $travel_to_ch_ncmp$ abstract event to the $travel_to_out_buf_ncmp$ and $travel_from_buf_to_ch_ncmp$ concrete events as shown in Fig. 18. The $travel_to_out_buf_ncmp$ event handles transferring of a waiting message from an input buffer to output buffer and the $travel_from_buf\ _to_ch_ncmp$ event deals with transferring of a waiting message from an output buffer to a channel. The same split is performed for $travel_to_ch_cmp$.

To transfer a message from an input buffer of a node to the selected output buffers, we define a local variable $avb_buf$ in $travel_to_out_buf_ncmp$ event. The $avb_buf$ variable contains the list of output buffers selected for transferring and with enough space to accept new messages (**@grd2**). We define the $avb_buf$ local variable in the $travel_to_out_buf_ncmp$ event because we need to exclude the selected output channels which are full and cannot receive any messages (**@grd1**) from all the selected output channels.

In **@act1** of $travel_to_out_buf_ncmp$, the message is added to the buffers listed in $avb_buf$. Moreover, the message waiting to be transferred to the output buffers, in $avb_buf$, is removed from the waiting list (**@act2**). We note that the message is not removed from the input buffer because there are instances of the message in the waiting list which are not transferred to output buffers. Removing the message from an input buffer of a node is done when the message is transferred to all the selected output buffers, in $waiting_inout_buf$ and is modelled by the $travel_to_out_buf_cmp$ event.

**travel_to_out_buf_ncmp**
**refines travel_to_ch_ncmp**
**any** $n$ $msg$ $avb_buf$
**where**
**@grd1** $\exists ch \cdot ch \in start_ch_nd^{-1}[\{n\}] \wedge ch \in waiting_inout_buf[\{msg\}]$
$\wedge card(buf_out(ch)) \geq bf_size$
**@grd2** $\forall c \cdot c \in start_ch_nd^{-1}[\{n\}] \wedge c \in waiting_inout_buf[\{msg\}]$
$\wedge card(buf_out(c)) < bf_size \Rightarrow c \in avb_buf$
**@grd3** $avb_buf \subseteq waiting_inout_buf[\{msg\}] \wedge avb_buf \subseteq start_ch_nd^{-1}[\{n\}]$
**then**
**@act1** $buf_out := buf_out \Leftarrow \{c \mapsto i \cup \{msg\} | c \in avb_buf \wedge i = buf_out(c)\}$
**@act2** $waiting_inout_buf := waiting_inout_buf \setminus (\{msg\} \times avb_buf)$
**end**

**travel_from_buf_to_ch_ncmp**
**refines travel_to_ch_ncmp**
**any** $n$ $msg$ $free_ch$
**where**
**@grd1** $\exists ch \cdot msg \in buf_out(ch) \wedge ch \in ran(mv_ch)$
**@grd2** $\forall ch \cdot ch \in ch_nd^{-1}[\{n\}] \wedge msg \in buf_out(ch) \wedge ch \notin ran(mv_ch) \Rightarrow ch \in free_ch$
**then**
**@act1** $mv_ch := mv_ch \cup (\{msg\} \times free_ch)$
**@act2** $buf_out := buf_out \Leftarrow \{c \mapsto i \setminus \{msg\} | c \mapsto i \in buf_out \wedge c \in free_ch\}$
**end**

Figure 18: The Machine Part 2 (Level 3): Events

Finally, the $travel_from_buf_to_ch_ncmp$ event models the message transferring from output buffers of a node to the channels. This event is enabled when there is a message in the output buffers of a node and the channels connected to the output buffers are free (**@grd2**). The message is removed from the output buffers (**@act2**) and added to the corresponding channels (**@act1**).

The statistics of proof obligations at this level of the framework's hierarchy are shown in Table 3.

Table 3: Proof Statistics Level 3

| Model | Number of Proof Obligations | Automatically Discharged | Interactively Discharged |
|---|---|---|---|
| *Level*3 Context | 0 | 0(0%) | 0(0%) |
| *Level*3 Machine | 120 | 80(67%) | 40(33%) |

The third level of the hierarchy presents the foundation framework of the network design. The development strategy of our framework put forwards the refinement method to be used for modelling and verifying specific net-

work designs. As demonstrated, in each hierarchical level we gradually add more elements and primitives together their corresponding properties to the abstract level until we achieve a more detailed model of the system. This method allows us to understand the problem from various facets, while we focus on the various parts of the system at each level.

We also note the integration aspect of our framwork. For instance, in Level 2 we add the `Channel` structural element to the network architecture model and at the same time, we need to develop the network primitives in terms of this element. The correctness of the algorithm-architecture integration is expressed by the model invariants that are proved correct for all the network primitives (i.e., algorithm) and elements (i.e., architecture).

### 6. A Theory Basis for the NetCorre framework

Based on the developed NetCorre framework put forward in the previous section, we now extract a network theory to be reused in concrete network modelling. NetCorre is independent of any design details and its components can be easily refined to concrete models until, e.g. a concrete routing algorithm is derived [13]. Each component (architectural or primitives) can be refined to different instances and the same instance of a component can be used in different network developments, e.g. to develop a FIFO-buffer [13]. Hence, it is imperative to raise the reusability of our framework to a more prominent level. For this, we build a library of pre-defined and pre-proved reusable units of the framework to be employed in various network design. This approach dramatically improves the lifecycle modelling and proving network designs. To produce a library of reusable units, we adapt the idea of object-oriented inheritance in which a root class is defined and then extended to other classes. Therefore, in this section the fundamental components of network design are defined as root classes by defining and proving theorems for these components. The introduced theorems are the *design rules* of our methodology; they can be extended to particular component developments and then used in refined models of the network.

In the rest of this section, we extend the mathematical language and theory of Event-B to specify the most relevant characteristics of network design in a network **extension**. This network extension encompasses definitions and properties from NetCorre that we deem fundamental for any network design. In other words, the proposed network extension consists of the NetCorre components that can be reused to specify and verify different

network designs. We introduce new predicate definitions, new operator definitions and theorems dedicated to network designs. The purpose of the new network extension is to avoid remodelling and especially reproving model properties every time we need a formal network design. Instead, we provide a pre-defined and pre-proved library containing fundamental characteristics of networked systems. We develop the network extension by using the *theory plug-in*. The proposed network extension provides a means for designers to develop various network models when the developments do not contradict the network theory.

We develop three theory levels, N1-N3, following the development order of NetCorre. Hence, the Event-B theory corresponding to the hierarchical framework is developed in a hierarchical manner as well. The overview of hierarchical development is shown in Fig. 19.

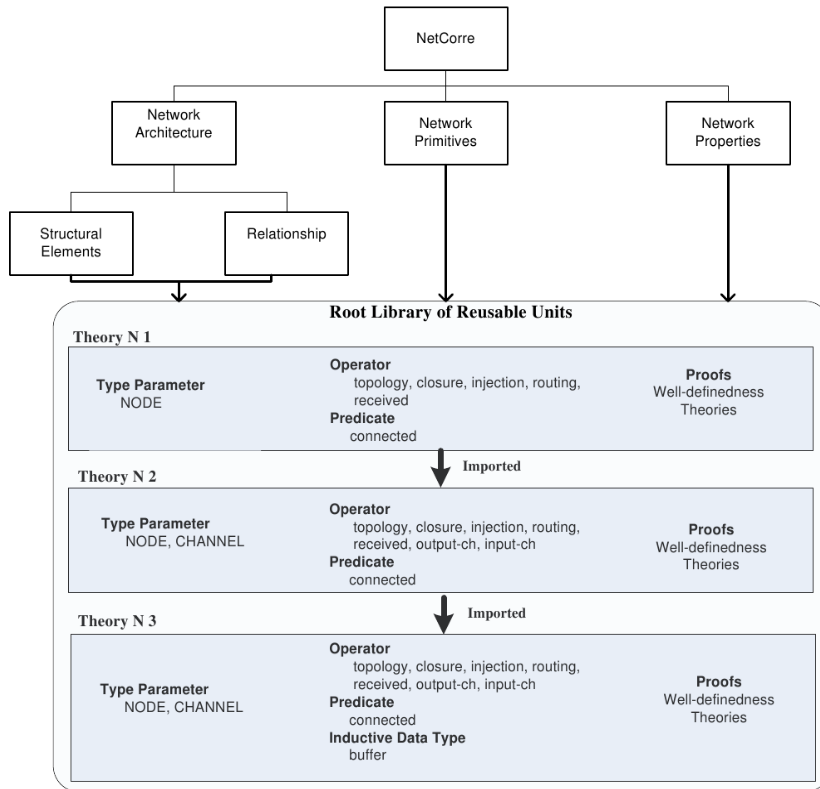



Figure 19: Network Theory of the Framework

*6.1. Theory N1 of the framework*

The first fundamental characteristic of any network is given by the *connectivity* of *nodes*, i.e., the topology of the network. This corresponds to the `Relationship` component of the framework. We define a *topology* operator where the interconnection of nodes in a network is shown. The *topology* operator takes a set $N$ and yields all the topologies whose pairs are in $N \times N$ and all the given nodes are connected via these pairs. The operator is a root class which can be instantiated to different types of topologies in the next levels. As shown in the following, *topology* is declared to be a prefix operator with a single argument represented by *N*. The well-definedness condition declares $N$ to be a subset of a global type parameter *NODE*. The *topology* is polymorphic on type *NODE* because *NODE* is a type parameter. The final clause defines the expression *topology(NODE)* in terms of the existing expression language and a new expression *connected*, explained shortly.

**operator** *topology*
**prefix**
**args** $N$
**condition** $N \subseteq NODE$
**definition** $\{Neigh \cdot Neigh \in N \leftrightarrow N \wedge Neigh = Neigh^{-1} \wedge dom(Neigh) \lhd id \cap Neigh = \varnothing$
$\wedge finite(Neigh) \wedge Neigh \neq \varnothing \wedge connected(N, Neigh) \mid Neigh\}$

The *connected* predicate is declared as prefix with two arguments $N$ and *net* as shown below. The condition specifies a well-definedness condition that $N$ should be a subset of *NODE* and *net* should be a set of pairs of nodes. The final clause provides the boolean value $TRUE$ indicating all the given nodes i.e., *N*, are connected via the given links, i.e, *net*. The main definition of *connected* relies on the existence of a link path between arbitrary nodes $n$ and $m$ from the given set *N*, i.e, the pair $n \mapsto m$ belongs to the transitive closure on the given links *net*. The clause defines the predicate *connected* in terms of a new expression *closure*, explained shortly.

| | |
|---|---|
| **predicate** | *connected* |
| **prefix** | |
| **args** | $N \quad net$ |
| **condition** | $N \subseteq NODE \land net \subseteq NODE \leftrightarrow NODE$ |
| **definition** | $\exists c \cdot c \in closure(net) \land (\forall n, m \cdot n \in N \land m \in N \Rightarrow n \mapsto m \in c)$ |

The *closure* operator is declared as prefix with an argument *net*. The condition specifies a well-definedness condition that *net* should be a set of pairs of nodes. The final clause provides the transitive closure of a binary relation on the set of pairs.

| | |
|---|---|
| **operator** | *closure* |
| **prefix** | |
| **args** | $net$ |
| **condition** | $net \subseteq NODE \leftrightarrow NODE$ |
| **definition** | $\{cl \cdot cl \in NODE \leftrightarrow NODE \land net \subseteq cl \land cl; net \subseteq cl$ <br> $\land (\forall s \cdot net \subseteq s \land s; net \subseteq s \Rightarrow cl \subseteq s) \mid cl\}$ |

Having introduced those new predicates and operators, we specify and prove theorems about them. The aim is to develop a library of rules derived from the introduced operators and to provide the possibility to prove any newly introduced theorem. The introduced theorems can then be used in the rule form in proofs.

The following theorems show the properties which should be preserved regarding the *connected* operator. The first theorem shows that an empty set of nodes is connected in any networks and the second one denotes that any subset of nodes connected in a sub-network are also connected on larger networks that include the sub-network. The last theorem denotes that when a set of nodes is connected in a network, any subsets of the set is also connected in the network. We note that any instances of the *topology* operator inherits all its proved theorems regarding connectivity.

**thm emptyset** $\forall L \cdot L \in NODE \leftrightarrow NODE \Rightarrow connected(\varnothing, L)$

**thm subnetCon** $\forall S, L, L2 \cdot L \in NODE \leftrightarrow NODE \wedge L2 \in NODE \leftrightarrow NODE$
$\wedge L \subseteq L2 \wedge connected(S, L) \Rightarrow connected(S, L2)$

**thm subnodeCon** $\forall S1, S2, L \cdot L \in NODE \leftrightarrow NODE \wedge S1 \in P(NODE)$
$\wedge S2 \in P(NODE) \wedge S1 \subseteq S2 \wedge connected(S2, L) \Rightarrow connected(S1, L)$

The abstract architecture component of the framework can be developed by applying these operators in the system development phase. The abstract `Network primitives` component of the framework consists of `Injecting`, `Routing` and `Receiving`; abstract operator for each of them is put forward in this section. The *injection* operator models the abstract behaviour of *injecting* sub-component of NetCorre by getting a list of messages and a network topology. The operator maps the list of messages to a number of nodes of a network. We note that we do not model details of messages such as source and destinations and these details can be defined later in a network model.

**operator** *injection*
**prefix**
**args** *msg* *net*
**condition** $net \in topology(NODE) \wedge msg \subseteq MSG$
**definition** $\{in \cdot in \in msg \rightarrow dom(net) \mid in\}$

The *routing* operator models the behaviour of the `routing` component in the framework by getting a list of messages with their location in the network and a network topology. For each message in the list, the operator either keeps the message to its current position or transmits it to neighbour nodes according to the topology given as an argument to the operator.

**operator** *routing*
**prefix**
**args** *cur_msg* *net*
**condition** $net \in topology(NODE) \wedge cur_msg \in MSG \leftrightarrow dom(net)$
**definition** $\{next \cdot next \in dom(cur_msg) \leftrightarrow dom(net) \wedge dom(next) = dom(cur_msg)$
$\wedge\ next \subseteq cur_msg; net \wedge next \neq cur_msg \mid next\}$

The *routing* operator is a reusable unit which acts as a root class in our development to instantiate different routing algorithms. The instantiated routing algorithms inherit all the proved properties of the *routing* operator and then they can be used in the development of particular networks.

The *received* operator is declared to be a prefix operator with two arguments: *msg* and *net*. The well-definedness conditions declare *msg* to be a subset of a global type parameter $MSG$ and *net* to be a relation, containing $NODE \leftrightarrow NODE$ pairs. The final clause defines the expression *received* in terms of the newly introduced network expression language.

**operator** *received*
**prefix**
**args** *msg* *net*
**condition** $net \in topology(NODE) \wedge msg \subseteq MSG$
**definition** $\{out \cdot out \in msg \leftrightarrow dom(net) \wedge dom(out) = msg$
$\wedge\ (\exists in.in \in injection(msg, net)) \mid out\}$

Having introduced the operators for the `network primitives`, we specify and prove theorems about their inter-relations. We develop a library of rules derived from the introduced operators to provide a means for users to verify the correctness of a network design by using these theorems in proofs. The first theorem denotes that no messages are lost during routing and the second one shows that for any set of messages injected to the network, there is the set of messages which are received by the nodes of the network.

**thm NoMsgLoss_in_Routing** $\forall net, next, m, in_m \cdot net \in topology(NODE)$
$\wedge\ in_m \in injection(MSG, net) \wedge m \in dom(in_m) \twoheadrightarrow dom(net)$
$\wedge\ next \in routing(net, m) \Rightarrow dom(next) = dom(m)$
**thm InjectionReceivedEquality** $\forall net, rcv, in_m \cdot net \in topology(NODE)$
$\wedge\ in_m \in injection(MSG, net) \wedge rcv \in received(dom(in_m), net)$
$\Rightarrow dom(rcv) = dom(in_m)$

### *6.2. Theory N2 of the framework*

The development of the theory follows the hierarchical development of the framework because the proposed hierarchy suits the construction of a

well-structured theory of the network. Therefore, to extend Level 1 of the hierarchical framework and model the `Channel` structural element, we create a new theory component N2 where N1 theory component is imported. The N2 theory component has a $CH$ global type parameter used in operator definitions. To express channels as a data transferring media, the global type parameter $CHANNEL$ should be connected to the topology concept defined by the *topology* operator. Channels connect neighbour nodes to each other, i.e., all pairs of nodes in any topology should be represented via channels. Therefore, two new operators *input_ch* and *output_ch* are introduced in N2. The *input_ch* operator is defined in the following way:

| **operator** *input_ch* |
|---|
| **prefix** |
| **args** $c$ $net$ |
| **condition** $c \subseteq CHANNEL \wedge net \in topology(NODE)$ |
| **definition** $\{sc \cdot sc \in c \twoheadrightarrow net \mid (sc; (net \lhd prj1))^{-1}\}$ |

The *input_ch* operator is declared as prefix with two arguments $c$ and $net$. The condition specifies a well-definedness condition that $c$ must be a subset of $CHANNEL$ and $net$ must be of type $topology(NODE)$. The final clause provides the definition of the expression *input_ch(CH, NODE ↔ NODE)* in terms of the N1 expression language. The definition declares mapping of channels to nodes that are connected to other nodes as shown in the *topology* definition.

For defining the *output_ch* operator we have three arguments $c$, $net$ and $sc$. The condition specifies that $c$ must be a subset of $CHANNEL$, $net$ must be of type $topology(NODE)$ and $sc$ must be of type $input_ch(c, net)$. The operator definition means that for any set of channels mapped to nodes by *input_ch* operator, a corresponding node to channel mapping is constructed in such a way that the composition of these two mapping is equal to the given topology.

**operator** $output_ch$

**prefix**

**args** $c$ $net$ $sc$

**condition** $c \subseteq CHANNEL \wedge net \in topology(NODE) \wedge sc \in input_ch(c, net)$

**definition** $\{ch_nd, ec \cdot ch_nd \in c \twoheadrightarrow net \wedge ec = ch_nd; (net \lhd prj1) \wedge sc; ec = net \mid ec^{-1}\}$

Besides, we specify theorems about `channel` structural elements and extend the proposed library of rules for network modelling. The first theorem shows that any channel is mapped to only one pair of nodes. In other words, the reverse of a derived relation from *output_ch* operator is a function. The second theorem denotes that given a network topology and a set of channels, the *input_ch* and *output_ch* operators can construct exactly the same topology plus more information about channels. This means that without considering channels in derived relations from *input_ch* and *output_ch*, we achieve the given topology. We note that both *input_ch* and *output_ch* are extensions of the *topology* operator in the N1 theory of the hierarchical framework. In other words, these two newly introduced operators are an implementation of the *topology* operator in the N1 theory.

**thm Rev_ch_fun** $\forall n, c, net, sc, ec \cdot n \subseteq NODE \wedge c \subseteq CHANNEL \wedge net \in topology(n)$
$\wedge\ sc \in input_ch(c, net) \wedge ec \in output_ch(c, net, sc) \Rightarrow ec \in n \leftrightarrow c \wedge ec^{-1} \in c \rightarrow n$

**thm chNet_eq_to_ch** $\forall n, c, net, sc, ec \cdot n \subseteq NODE \wedge c \subseteq CHANNEL \wedge net \in topology(n)$
$\wedge\ sc \in input_ch(c, net) \wedge ec \in output_ch(c, net, sc) \Rightarrow sc; ec^{-1} = net$

### *6.3. Theory N3 of the framework*

The third level of the hierarchical framework NetCorre introduces the $buffer$ structural element; correspondingly, we create a new theory component to define rules about buffers where the N2 theory component is imported. We define a new inductive datatype to represent buffers; this can then be used to define the *enqueue* and *dequeue* operators for specific types of buffers. The buffer constructors consist *nil* and *cons* that seperate forms that an element might take. The *cons* constructor composed of two destructors of *head* and *tail* that provide a means of accessing the feild of an element.

**datatypes** $buffer$
**type arguments** $MESSAGE$
**constructors**
$nil$
$cons$
**destructors**
$head$ **type** $MESSAGE$
$tail$ **type** $buffer(MESSAGE)$

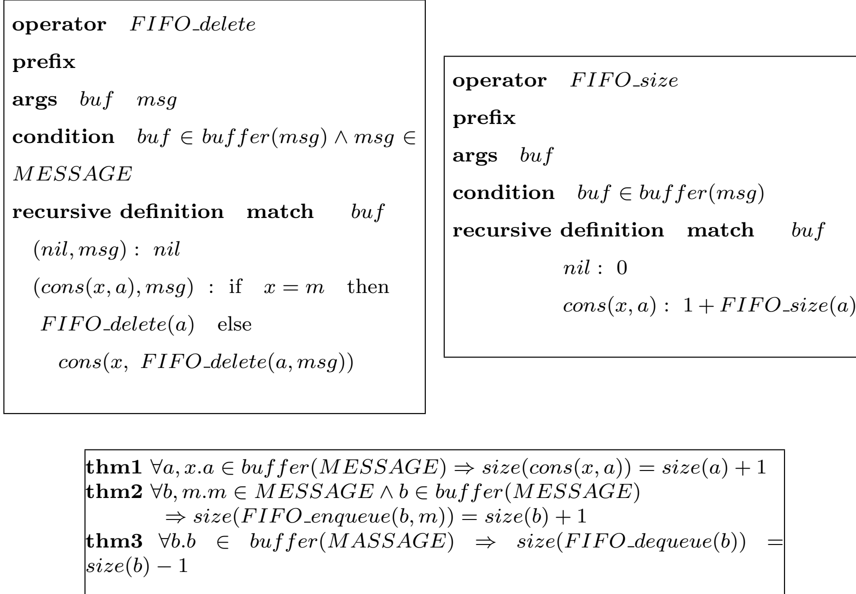
*6.4. Using the Theory of the framework*

These three network theory levels form the basis of design rules in our methodology and any extension of these rules adds an instance of the network components to the library of design rules. As an example of extending the theory Level N3, the *enqueue* and *dequeue* operators of a FIFO-buffer are illustrated in the following.

The *enqueue* operator is declared to be a prefix operator with two arguments, $buf$ and $msg$; it models the adding of the new message $msg$ to the top of the queue $buf$. The *dequeue* operator is defined to be a prefix operator with the argument $buf$ and models the removal of the message at the bottom of the queue $buf$. The functioning of the operators is ensured with the theorems **thm1**, **thm2**, and **thm3**.

**operator** $FIFO_enqueue$
**prefix**
**args** $buf$ $msg$
**condition** $buf \in buffer(msg)$
$\wedge\ msg \in MESSAGE$
**definition** $cons(m, b)$

**operator** $FIFO_dequeue$
**prefix**
**args** $buf$
**condition** $buf \in buffer(msg)$
$\wedge\ buf \neq nil$
**recursive definition** **match** $buf$
$nil:\ b$
$cons(x, a):\ if\ a =\ nil\ then\ nil$
$else\ cons(x,\ FIFO_dequeue\ (a, msg))$




In Fig. 20 we point out the 'position' in the hierarchical development that the *enqueue* and *dequeue* operators for the FIFO buffer have. Other extensions can take place - we have depicted for the sake of example a 'Minimal Routing' theory in Fig. 20. In fact, any of the N1-N3 theory levels can be extended, as well as the newly developed ones such as the **FIFO-buffer**.

## 7. Conclusion

In this paper, we have introduced the hierarchical framework NetCorre for correct network design, we have modeled it formally, and we have proven various properties about it. The instrumental formalism and tool support in our modelling and proving endeavors is provided by the Event-B formal method and its associated Rodin platform. The formal design technique of refinement is at the centre of all our developments in this paper. It is through refinement that we are able to model a hierarchical framework and also through refinement that the intended reuse of NetCorre is put forward. A network specification based on NetCorre can be refined until a network implementation is derived; some examples of derivations appear in [13]. Based on NetCorre, we have defined and proved some fundamental 'design rules' for a network theory developed within the theory plugin of the Rodin toolset. These design rules can then be used to develop even more concrete design

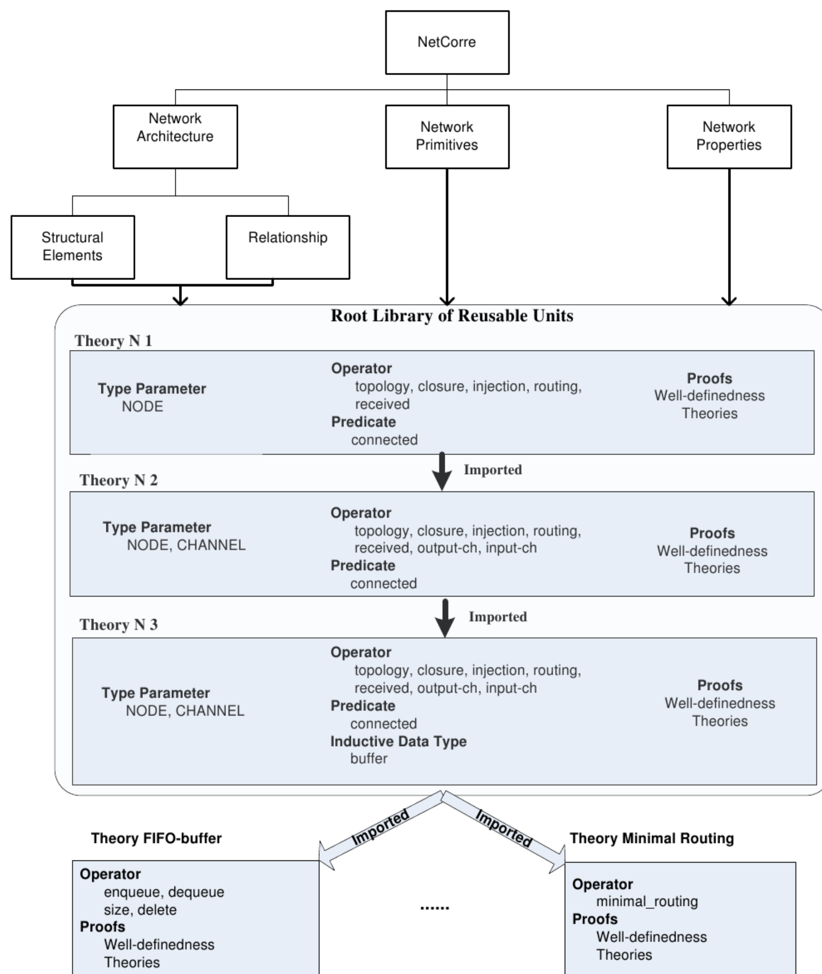



Figure 20: Instantiation of Network Theory

rules, where precise definitions are proposed for the functional algorithms or architecture components.

We thus demonstrate that a systematic and effective specification and verification of various network designs is possible, based upon a small set of proved structural and functional network primitives, that are added or detailed gradually. The framework allows to make a variety of design choices in the early stages of the design with various degrees of modelling details, relying on the formal refinement technique. Our framework integrates both the models for the algorithm and the architecture network design so that the overall correctness is guaranteed.

We observe that our proposed NetCorre is very general, so that it can be reused to model a wide variety of network designs. Remarkably, both networks 'in the large' as well as networks-on-chip are approachable via our framework, because defining NetCorre is based on previously engineering numerous networking models in Event-B [12, 14]. With NetCorre, we extract the most general reusable network characteristics and put them up for further reuse.

## References


[1] Abrial, J.: A System Development Process with Event-B and the Rodin Platform. chap. 1, pp. 1–3. Springer Berlin / Heidelberg, Berlin, Heidelberg (2007). DOI http://dx.doi.org/10.1007/978-3-540-76650-6_1. URL http://dx.doi.org/10.1007/978-3-540-76650-6_1

[2] Abrial, J.: A system development process with event-b and the rodin platform. Formal Methods and Software Engineering pp. 1–3 (2007)

[3] Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)

[4] Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in event-b. Int. J. Softw. Tools Technol. Transf. **12**, 447–466 (2010). DOI http://dx.doi.org/10.1007/s10009-010-0145-y. URL http://dx.doi.org/10.1007/s10009-010-0145-y

[5] Abrial, J.R., Butler, M., Hallerstede, S., Leuschel, M., Schmalz, M., Voisin, L.: Proposals for mathematical extensions for event-b. Technical Report

[6] Attiogbé, C.: Event-based approach to modelling dynamic architecture: Application to mobile ad-hoc network. In: ISoLA, pp. 769–781 (2008)

[7] Attiogbé, C.: Modelling and analysing dynamic decentralised systems. CoRR **abs/0909.4896** (2009)

[8] Back, R.J.R., Kurki-Suonio, R.: Decentralization of process nets with centralized control. In: Proceedings of the second annual ACM symposium on Principles of distributed computing, PODC '83, pp. 131–142. ACM, New York, NY, USA (1983). DOI 10.1145/800221.806716. URL http://doi.acm.org/10.1145/800221.806716

[9] Butler, M., Maamria, I.: Mathematical extension in event-b through the rodin theory component. Technical Report

[10] Cansell, D., Méry, D., Proch, C.: System-on-chip design by proof-based refinement. Int. J. Softw. Tools Technol. Transf. **11**(3), 217–238 (2009). DOI 10.1007/s10009-009-0104-7. URL http://dx.doi.org/10.1007/s10009-009-0104-7

[11] Hoang, T.S., Kuruma, H., Basin, D.A., Abrial, J.R.: Developing topology discovery in event-b. In: IFM, pp. 1–19 (2009)

[12] Kamali, M., Laibinis, L., Petre, L., Sere, K.: Formal development of wireless sensoractor networks. Science of Computer Programming (0) (2012). DOI 10.1016/j.scico.2012.03.002. URL http://www.sciencedirect.com/science/article/pii/S0167642312000470?v=s5

[13] Kamali, M., Petre, L., Sere, K., Daneshtalab, M.: Correcomm: A formal hierarchical framework for communication designs. In: the 2nd IEEE International Conference on Embedded Systems for Enterprise Applications. IEEE (2011)

[14] Kamali, M., Petre, L., Sere, K., Daneshtalab, M.: Formal modeling of multicast communication in 3d nocs. In: Digital System Design (DSD), 2011 14th Euromicro Conference on, pp. 634 –642 (2011). DOI 10.1109/DSD.2011.86

[15] Kamali, M., Petre, L., Sere, K., Daneshtalab, M.: Refinement-based modeling of 3d nocs. In: Fundamental of Software Engineering (FSEN),

2011 4th IPM International Conference on, vol. 7141, pp. 236 –252. Springer-Verlag (2012)

[16] Lucarz, C., Mattavelli, M., Dubois, J.: A co-design platform for algorithm/architecture design exploration. In: Multimedia and Expo, 2008 IEEE International Conference on, pp. 1069 –1072 (2008). DOI 10.1109/ICME.2008.4607623

[17] Meng, Y.: Algorithm/architecture design space co-exploration for energy efficient wireless communications systems. Ph.D. thesis, Santa Barbara, CA, USA (2006). AAI3233117

[18] Métayer, C., Voisin, L.: The Event-B Mathematical Language (Version 2) (2009). URL http://deploy-eprints.ecs.soton.ac.uk/11/4/kernel_lang.pdf

[19] Plosila, J., Sere, K.: Action systems in pipelined processor design. In: In Proc. of the 3rd Int. Symp. on Advanced Research in Asynchronous Circuits and Systems, pp. 156–166. IEEE Computer Society Press (1997)

[20] Rigorous Open Development Environment for Complex Systems (RODIN), IST FP6 STREP project [online]: Documentation at http://rodin.cs.ncl.ac.uk/ (accessed 26 october 2011)

[21] RODIN tool platform [online]: Available at http://www.event-b.org/platform.html/ (Accessed 26 October 2011)

[22] Zhu, X., Malik, S.: A hierarchical modeling framework for on-chip communication architectures of multiprocessing socs. ACM Trans. Des. Autom. Electron. Syst. **12**(1), 6:1–6:24 (2007). DOI 10.1145/1188275.1188281. URL http://doi.acm.org/10.1145/1188275.1188281

# Turku Centre for Computer Science
# TUCS Dissertations

1. **Marjo Lipponen**, On Primitive Solutions of the Post Correspondence Problem
2. **Timo Käkölä**, Dual Information Systems in Hyperknowledge Organizations
3. **Ville Leppänen**, Studies on the Realization of PRAM
4. **Cunsheng Ding**, Cryptographic Counter Generators
5. **Sami Viitanen**, Some New Global Optimization Algorithms
6. **Tapio Salakoski**, Representative Classification of Protein Structures
7. **Thomas Långbacka**, An Interactive Environment Supporting the Development of Formally Correct Programs
8. **Thomas Finne**, A Decision Support System for Improving Information Security
9. **Valeria Mihalache**, Cooperation, Communication, Control. Investigations on Grammar Systems.
10. **Marina Waldén**, Formal Reasoning About Distributed Algorithms
11. **Tero Laihonen**, Estimates on the Covering Radius When the Dual Distance is Known
12. **Lucian Ilie**, Decision Problems on Orders of Words
13. **Jukkapekka Hekanaho**, An Evolutionary Approach to Concept Learning
14. **Jouni Järvinen**, Knowledge Representation and Rough Sets
15. **Tomi Pasanen**, In-Place Algorithms for Sorting Problems
16. **Mika Johnsson**, Operational and Tactical Level Optimization in Printed Circuit Board Assembly
17. **Mats Aspnäs**, Multiprocessor Architecture and Programming: The Hathi-2 System
18. **Anna Mikhajlova**, Ensuring Correctness of Object and Component Systems
19. **Vesa Torvinen**, Construction and Evaluation of the Labour Game Method
20. **Jorma Boberg**, Cluster Analysis. A Mathematical Approach with Applications to Protein Structures
21. **Leonid Mikhajlov**, Software Reuse Mechanisms and Techniques: Safety Versus Flexibility
22. **Timo Kaukoranta**, Iterative and Hierarchical Methods for Codebook Generation in Vector Quantization
23. **Gábor Magyar**, On Solution Approaches for Some Industrially Motivated Combinatorial Optimization Problems
24. **Linas Laibinis**, Mechanised Formal Reasoning About Modular Programs
25. **Shuhua Liu**, Improving Executive Support in Strategic Scanning with Software Agent Systems
26. **Jaakko Järvi**, New Techniques in Generic Programming – C++ is more Intentional than Intended
27. **Jan-Christian Lehtinen**, Reproducing Kernel Splines in the Analysis of Medical Data
28. **Martin Büchi**, Safe Language Mechanisms for Modularization and Concurrency
29. **Elena Troubitsyna**, Stepwise Development of Dependable Systems
30. **Janne Näppi**, Computer-Assisted Diagnosis of Breast Calcifications
31. **Jianming Liang**, Dynamic Chest Images Analysis
32. **Tiberiu Seceleanu**, Systematic Design of Synchronous Digital Circuits
33. **Tero Aittokallio**, Characterization and Modelling of the Cardiorespiratory System in Sleep-Disordered Breathing
34. **Ivan Porres**, Modeling and Analyzing Software Behavior in UML
35. **Mauno Rönkkö**, Stepwise Development of Hybrid Systems
36. **Jouni Smed**, Production Planning in Printed Circuit Board Assembly
37. **Vesa Halava**, The Post Correspondence Problem for Market Morphisms
38. **Ion Petre**, Commutation Problems on Sets of Words and Formal Power Series
39. **Vladimir Kvassov**, Information Technology and the Productivity of Managerial Work
40. **Frank Tétard**, Managers, Fragmentation of Working Time, and Information Systems

**41.** **Jan Manuch**, Defect Theorems and Infinite Words
**42.** **Kalle Ranto**, $Z_4$-Goethals Codes, Decoding and Designs
**43.** **Arto Lepistö**, On Relations Between Local and Global Periodicity
**44.** **Mika Hirvensalo**, Studies on Boolean Functions Related to Quantum Computing
**45.** **Pentti Virtanen**, Measuring and Improving Component-Based Software Development
**46.** **Adekunle Okunoye**, Knowledge Management and Global Diversity – A Framework to Support Organisations in Developing Countries
**47.** **Antonina Kloptchenko**, Text Mining Based on the Prototype Matching Method
**48.** **Juha Kivijärvi**, Optimization Methods for Clustering
**49.** **Rimvydas Rukšėnas**, Formal Development of Concurrent Components
**50.** **Dirk Nowotka**, Periodicity and Unbordered Factors of Words
**51.** **Attila Gyenesei**, Discovering Frequent Fuzzy Patterns in Relations of Quantitative Attributes
**52.** **Petteri Kaitovaara**, Packaging of IT Services – Conceptual and Empirical Studies
**53.** **Petri Rosendahl**, Niho Type Cross-Correlation Functions and Related Equations
**54.** **Péter Majlender**, A Normative Approach to Possibility Theory and Soft Decision Support
**55.** **Seppo Virtanen**, A Framework for Rapid Design and Evaluation of Protocol Processors
**56.** **Tomas Eklund**, The Self-Organizing Map in Financial Benchmarking
**57.** **Mikael Collan**, Giga-Investments: Modelling the Valuation of Very Large Industrial Real Investments
**58.** **Dag Björklund**, A Kernel Language for Unified Code Synthesis
**59.** **Shengnan Han**, Understanding User Adoption of Mobile Technology: Focusing on Physicians in Finland
**60.** **Irina Georgescu**, Rational Choice and Revealed Preference: A Fuzzy Approach
**61.** **Ping Yan**, Limit Cycles for Generalized Liénard-Type and Lotka-Volterra Systems
**62.** **Joonas Lehtinen**, Coding of Wavelet-Transformed Images
**63.** **Tommi Meskanen,** On the NTRU Cryptosystem
**64.** **Saeed Salehi**, Varieties of Tree Languages
**65.** **Jukka Arvo**, Efficient Algorithms for Hardware-Accelerated Shadow Computation
**66.** **Mika Hirvikorpi**, On the Tactical Level Production Planning in Flexible Manufacturing Systems
**67.** **Adrian Costea**, Computational Intelligence Methods for Quantitative Data Mining
**68.** **Cristina Seceleanu**, A Methodology for Constructing Correct Reactive Systems
**69.** **Luigia Petre**, Modeling with Action Systems
**70.** **Lu Yan**, Systematic Design of Ubiquitous Systems
**71.** **Mehran Gomari**, On the Generalization Ability of Bayesian Neural Networks
**72.** **Ville Harkke**, Knowledge Freedom for Medical Professionals – An Evaluation Study of a Mobile Information System for Physicians in Finland
**73.** **Marius Cosmin Codrea**, Pattern Analysis of Chlorophyll Fluorescence Signals
**74.** **Aiying Rong**, Cogeneration Planning Under the Deregulated Power Market and Emissions Trading Scheme
**75.** **Chihab BenMoussa**, Supporting the Sales Force through Mobile Information and Communication Technologies: Focusing on the Pharmaceutical Sales Force
**76.** **Jussi Salmi**, Improving Data Analysis in Proteomics
**77.** **Orieta Celiku**, Mechanized Reasoning for Dually-Nondeterministic and Probabilistic Programs
**78.** **Kaj-Mikael Björk**, Supply Chain Efficiency with Some Forest Industry Improvements
**79.** **Viorel Preoteasa**, Program Variables – The Core of Mechanical Reasoning about Imperative Programs
**80.** **Jonne Poikonen**, Absolute Value Extraction and Order Statistic Filtering for a Mixed-Mode Array Image Processor
**81.** **Luka Milovanov**, Agile Software Development in an Academic Environment
**82.** **Francisco Augusto Alcaraz Garcia**, Real Options, Default Risk and Soft Applications
**83.** **Kai K. Kimppa**, Problems with the Justification of Intellectual Property Rights in Relation to Software and Other Digitally Distributable Media
**84.** **Dragoş Truşcan**, Model Driven Development of Programmable Architectures
**85.** **Eugen Czeizler**, The Inverse Neighborhood Problem and Applications of Welch Sets in Automata Theory

**86.** **Sanna Ranto**, Identifying and Locating-Dominating Codes in Binary Hamming Spaces
**87.** **Tuomas Hakkarainen**, On the Computation of the Class Numbers of Real Abelian Fields
**88.** **Elena Czeizler**, Intricacies of Word Equations
**89.** **Marcus Alanen**, A Metamodeling Framework for Software Engineering
**90.** **Filip Ginter**, Towards Information Extraction in the Biomedical Domain: Methods and Resources
**91.** **Jarkko Paavola**, Signature Ensembles and Receiver Structures for Oversaturated Synchronous DS-CDMA Systems
**92.** **Arho Virkki**, The Human Respiratory System: Modelling, Analysis and Control
**93.** **Olli Luoma**, Efficient Methods for Storing and Querying XML Data with Relational Databases
**94.** **Dubravka Ilić**, Formal Reasoning about Dependability in Model-Driven Development
**95.** **Kim Solin**, Abstract Algebra of Program Refinement
**96.** **Tomi Westerlund**, Time Aware Modelling and Analysis of Systems-on-Chip
**97.** **Kalle Saari**, On the Frequency and Periodicity of Infinite Words
**98.** **Tomi Kärki**, Similarity Relations on Words: Relational Codes and Periods
**99.** **Markus M. Mäkelä**, Essays on Software Product Development: A Strategic Management Viewpoint
**100.** **Roope Vehkalahti**, Class Field Theoretic Methods in the Design of Lattice Signal Constellations
**101.** **Anne-Maria Ernvall-Hytönen**, On Short Exponential Sums Involving Fourier Coefficients of Holomorphic Cusp Forms
**102.** **Chang Li**, Parallelism and Complexity in Gene Assembly
**103.** **Tapio Pahikkala**, New Kernel Functions and Learning Methods for Text and Data Mining
**104.** **Denis Shestakov**, Search Interfaces on the Web: Querying and Characterizing
**105.** **Sampo Pyysalo**, A Dependency Parsing Approach to Biomedical Text Mining
**106.** **Anna Sell**, Mobile Digital Calendars in Knowledge Work
**107.** **Dorina Marghescu**, Evaluating Multidimensional Visualization Techniques in Data Mining Tasks
**108.** **Tero Säntti**, A Co-Processor Approach for Efficient Java Execution in Embedded Systems
**109.** **Kari Salonen**, Setup Optimization in High-Mix Surface Mount PCB Assembly
**110.** **Pontus Boström**, Formal Design and Verification of Systems Using Domain-Specific Languages
**111.** **Camilla J. Hollanti**, Order-Theoretic Mehtods for Space-Time Coding: Symmetric and Asymmetric Designs
**112.** **Heidi Himmanen**, On Transmission System Design for Wireless Broadcasting
**113.** **Sébastien Lafond**, Simulation of Embedded Systems for Energy Consumption Estimation
**114.** **Evgeni Tsivtsivadze**, Learning Preferences with Kernel-Based Methods
**115.** **Petri Salmela**, On Commutation and Conjugacy of Rational Languages and the Fixed Point Method
**116.** **Siamak Taati**, Conservation Laws in Cellular Automata
**117.** **Vladimir Rogojin**, Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation
**118.** **Alexey Dudkov**, Chip and Signature Interleaving in DS CDMA Systems
**119.** **Janne Savela**, Role of Selected Spectral Attributes in the Perception of Synthetic Vowels
**120.** **Kristian Nybom**, Low-Density Parity-Check Codes for Wireless Datacast Networks
**121.** **Johanna Tuominen**, Formal Power Analysis of Systems-on-Chip
**122.** **Teijo Lehtonen**, On Fault Tolerance Methods for Networks-on-Chip
**123.** **Eeva Suvitie**, On Inner Products Involving Holomorphic Cusp Forms and Maass Forms
**124.** **Linda Mannila**, Teaching Mathematics and Programming – New Approaches with Empirical Evaluation
**125.** **Hanna Suominen**, Machine Learning and Clinical Text: Supporting Health Information Flow
**126.** **Tuomo Saarni**, Segmental Durations of Speech
**127.** **Johannes Eriksson**, Tool-Supported Invariant-Based Programming

**128.** **Tero Jokela**, Design and Analysis of Forward Error Control Coding and Signaling for Guaranteeing QoS in Wireless Broadcast Systems
**129.** **Ville Lukkarila**, On Undecidable Dynamical Properties of Reversible One-Dimensional Cellular Automata
**130.** **Qaisar Ahmad Malik**, Combining Model-Based Testing and Stepwise Formal Development
**131.** **Mikko-Jussi Laakso**, Promoting Programming Learning: Engagement, Automatic Assessment with Immediate Feedback in Visualizations
**132.** **Riikka Vuokko**, A Practice Perspective on Organizational Implementation of Information Technology
**133.** **Jeanette Heidenberg**, Towards Increased Productivity and Quality in Software Development Using Agile, Lean and Collaborative Approaches
**134.** **Yong Liu**, Solving the Puzzle of Mobile Learning Adoption
**135.** **Stina Ojala**, Towards an Integrative Information Society: Studies on Individuality in Speech and Sign
**136.** **Matteo Brunelli**, Some Advances in Mathematical Models for Preference Relations
**137.** **Ville Junnila**, On Identifying and Locating-Dominating Codes
**138.** **Andrzej Mizera**, Methods for Construction and Analysis of Computational Models in Systems Biology. Applications to the Modelling of the Heat Shock Response and the Self-Assembly of Intermediate Filaments.
**139.** **Csaba Ráduly-Baka**, Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments
**140.** **Jari Kyngäs**, Solving Challenging Real-World Scheduling Problems
**141.** **Arho Suominen**, Notes on Emerging Technologies
**142.** **József Mezei**, A Quantitative View on Fuzzy Numbers
**143.** **Marta Olszewska**, On the Impact of Rigorous Approaches on the Quality of Development
**144.** **Antti Airola**, Kernel-Based Ranking: Methods for Learning and Performace Estimation
**145.** **Aleksi Saarela**, Word Equations and Related Topics: Independence, Decidability and Characterizations
**146.** **Lasse Bergroth**, Kahden merkkijonon pisimmän yhteisen alijonon ongelma ja sen ratkaiseminen
**147.** **Thomas Canhao Xu**, Hardware/Software Co-Design for Multicore Architectures
**148.** **Tuomas Mäkilä**, Software Development Process Modeling – Developers Perspective to Contemporary Modeling Techniques
**149.** **Shahrokh Nikou**, Opening the Black-Box of IT Artifacts: Looking into Mobile Service Characteristics and Individual Perception
**150.** **Alessandro Buoni**, Fraud Detection in the Banking Sector: A Multi-Agent Approach
**151.** **Mats Neovius**, Trustworthy Context Dependency in Ubiquitous Systems
**152.** **Fredrik Degerlund**, Scheduling of Guarded Command Based Models
**153.** **Amir-Mohammad Rahmani-Sane**, Exploration and Design of Power-Efficient Networked Many-Core Systems
**154.** **Ville Rantala**, On Dynamic Monitoring Methods for Networks-on-Chip
**155.** **Mikko Pelto**, On Identifying and Locating-Dominating Codes in the Infinite King Grid
**156.** **Anton Tarasyuk**, Formal Development and Quantitative Verification of Dependable Systems
**157.** **Muhammad Mohsin Saleemi**, Towards Combining Interactive Mobile TV and Smart Spaces: Architectures, Tools and Application Development
**158.** **Tommi J. M. Lehtinen**, Numbers and Languages
**159.** **Peter Sarlin**, Mapping Financial Stability
**160.** **Alexander Wei Yin**, On Energy Efficient Computing Platforms
**161.** **Mikołaj Olszewski**, Scaling Up Stepwise Feature Introduction to Construction of Large Software Systems
**162.** **Maryam Kamali**, Reusable Formal Architectures for Networked Systems

Turku Centre *for* Computer Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www. tucs.fi

**University of Turku**
*Faculty of Mathematics and Natural Sciences*
- Department of Information Technology
- Department of Mathematics and Statistics

*Turku School of Economics*
- Institute of Information Systems Science

**Åbo Akademi University**
*Division for Natural Sciences and Technology*
- Department of Information Technologies

ISBN 978-952-12-2932-9
ISSN 1239-1883

Maryam Kamali

# Reusable Formal Architectures for Networked Systems

Maryam Kamali

Reusable Formal Architectures for Networked Systems