Petter Sandvik

Formal Modelling for Digital Media Distribution

Turku Centre *for* Computer Science

TUCS Dissertations
No 206, November 2015

Formal Modelling for Digital Media Distribution

Petter Sandvik

To be presented, with the permission of the Faculty of Science and Engineering at Åbo Akademi University, for public criticism in Auditorium Gamma on November 13, 2015, at 12 noon.

Åbo Akademi University
Faculty of Science and Engineering
Joukahainengatan 3-5 A, 20520 Åbo, Finland

2015

Supervisors

Associate Professor Luigia Petre
Faculty of Science and Engineering
Åbo Akademi University
Joukahainengatan 3-5 A, 20520 Åbo
Finland

Professor Kaisa Sere
Faculty of Science and Engineering
Åbo Akademi University
Joukahainengatan 3-5 A, 20520 Åbo
Finland

Reviewers

Professor Michael Butler
Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton
Highfield, Southampton SO17 1BJ
United Kingdom

Professor Gheorghe Ștefănescu
Computer Science Department
University of Bucharest
14 Academiei Str., Bucharest, RO-010014
Romania

Opponent

Professor Michael Butler
Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton
Highfield, Southampton SO17 1BJ
United Kingdom

ISBN 978-952-12-3294-7
ISSN 1239-1883

To those who are no longer with us,
and to those who will be here after we are gone.

Abstract

Human beings have always strived to preserve their memories and spread their ideas. In the beginning this was always done through human interpretations, such as telling stories and creating sculptures. Later, technological progress made it possible to create a recording of a phenomenon; first as an analogue recording onto a physical object, and later digitally, as a sequence of bits to be interpreted by a computer. By the end of the 20th century technological advances had made it feasible to distribute media content over a computer network instead of on physical objects, thus enabling the concept of digital media distribution.

Many digital media distribution systems already exist, and their continued, and in many cases increasing, usage is an indicator for the high interest in their future enhancements and enriching. By looking at these digital media distribution systems, we have identified three main areas of possible improvement: network structure and coordination, transport of content over the network, and the encoding used for the content. In this thesis, our aim is to show that improvements in performance, efficiency and availability can be done in conjunction with improvements in software quality and reliability through the use of formal methods: mathematical approaches to reasoning about software so that we can prove its correctness, together with the desirable properties.

We envision a complete media distribution system based on a distributed architecture, such as peer-to-peer networking, in which different parts of the system have been formally modelled and verified. Starting with the network itself, we show how it can be formally constructed and modularised in the Event-B formalism, such that we can separate the modelling of one node from the modelling of the network itself. We also show how the piece selection algorithm in the BitTorrent peer-to-peer transfer protocol can be adapted for on-demand media streaming, and how this can be modelled in Event-B. Furthermore, we show how modelling one peer in Event-B can give results similar to simulating an entire network of peers.

Going further, we introduce a formal specification language for content transfer algorithms, and show that having such a language can make these algorithms easier to understand. We also show how generating Event-B code

from this language can result in less complexity compared to creating the models from written specifications. We also consider the decoding part of a media distribution system by showing how video decoding can be done in parallel. This is based on formally defined dependencies between frames and blocks in a video sequence; we have shown that also this step can be performed in a way that is mathematically proven correct.

Our modelling and proving in this thesis is, in its majority, tool-based. This provides a demonstration of the advance of formal methods as well as their increased reliability, and thus, advocates for their more wide-spread usage in the future.

Sammanfattning

Människor har alltid strävat efter att bevara sina minnen och sprida sina idéer. Till en början gjordes detta alltid genom mänskliga tolkningar, såsom historieberättande eller skapande av skulpturer. Senare gjorde tekniska framsteg det möjligt att göra inspelningar av fenomen; först som en analog inspelning på ett fysiskt föremål, och senare digitalt, som en sekvens bitar att tolkas av en dator. Vid senare halvan av 1990-talet hade tekniken gått så långt framåt att det var realistiskt att distribuera audiovisuell media över ett datornätverk istället för på fysiska föremål, och därmed skapades konceptet digital mediadistribution.

Ett flertal digitala mediadistributionssystem existerar redan, och deras fortsatta, och i många fall ökande användning indikerar att det finns ett stort intresse för framtida vidareutveckling och utökning av dylika system. Genom att se på dessa digitala mediadistributionssystem har vi identifierat tre tänkbara områden för förbättring: nätverksstruktur och -koordination, överföring av data över nätverket, och kodningen som används för innehållet. I denna avhandling vill vi visa att förbättringar i prestanda, effektivitet och tillgänglighet kan göras i samförstånd med förbättringar i programvarans kvalitet och pålitlighet genom att använda formella metoder: matematiska tillvägagångssätt för att resonera om mjukvara på så sätt att vi kan bevisa dess korrekthet tillsammans med andra önskvärda egenskaper.

Vi föreställer oss ett komplett mediadistributionssystem baserat på en distribuerad arkitektur, såsom peer-to-peer-nätverk, där olika delar av systemet har formellt modellerats och verifierats. Börjandes från själva nätverket visar vi hur det kan bli formellt konstruerat och modulariserat i Event-B-formalismen, så att vi kan separera modellerandet av en nod från modellerandet av nätverket självt. Vi visar också hur den algoritm som används för att välja vilken del av innehållet som ska överföras till näst i BitTorrent-filöverföringsprotokollet kan anpassas för strömmande media, och hur detta kan modelleras i Event-B. Vidare visar vi hur modellerande av en nod i Event-B kan ge liknande resultat som att simulera ett helt nätverk av noder.

Vi fortsätter genom att introducera ett formellt specifikationsspråk för dataöverföringsalgoritmer, och visar att genom att ha ett sådant språk kan dessa algoritmer bli enklare att förstå. Vi visar också hur vi kan minska

komplexiteten hos våra modeller genom att generera Event-B-kod från vårt formella specifikationsspråk istället för direkt från skrivna specifikationer. Vårt arbete berör också avkodningssidan av ett mediadistributionssystem genom att visa hur videoavkodning kan ske parallellt. Detta baserar sig på formellt definierade beroenden mellan ramar och block i en videosekvens; vi har visat att även detta steg kan utföras på ett sätt vars korrekthet har bevisats matematiskt.

Modelleringen och bevisningen i denna avhandling är huvudsakligen verktygsbaserad. Detta demonstrerar hur formella metoder har framskridit och blivit mer pålitliga, vilket också talar för att deras användning kan få en vidare utbredning i framtiden.

Acknowledgements

First and foremost, I would like to thank my supervisor, Associate Prof. Luigia Petre, with whom I have had the fortune of working together on several different topics during the last few years. I am also very grateful to my other supervisor, the late Prof. Kaisa Sere, who was supportive of my ideas already when I was a master student, and encouraged me to continue towards a PhD. Many thanks also to my opponent, Prof. Michael Butler, for taking the time to thoroughly review my thesis and be the opponent at my defence. Likewise, I am very grateful to Prof. Gheorghe Ștefănescu for his review and constructive criticism of my thesis.

I am lucky to have had many wonderful coworkers during my time as a PhD student. Among these are my closest coworkers currently or previously belonging to the Formal Methods and Networks group of the Distributed Systems Lab: Mats Neovius, Maryam Kamali, Mojgan Kamali, as well as Luigia Petre. I am also grateful to Marina Waldén who, among other things, helped me get started with formal modelling. Furthermore I would like to thank Marta Olszewska and Yuliya Prokhorova, both of whom I have been fortunate enough to have as office mates, as well as the other current and former members of the Distributed Systems Lab I have had the pleasure of working with: Pontus Boström, Jonatan Wiik, Fredrik Degerlund, Sergey Ostroumov, Linas Laibinis, Elena Troubitsyna, Anton Tarasyuk, Inna Pereverzeva, Johan Ersfolk, Andrew Edmunds, and Leonidas Tsiopoulos. For improving my time as a PhD student at Åbo Akademi and Turku Centre for Computer Science I am also thankful to the following people: Ion Petre, Bogdan Iancu, Cristian Gratie, Johannes Eriksson, Susanne Ramstedt, Nina Hultholm, Tove Österroos, Christel Engblom, and Tomi Suovuo. I would also like to thank my other co-authors and collaborators: Kristian Lumme and Erik Lumme; Mauno Rönkkö, Mikko Kolehmainen and Markus Stocker at University of Eastern Finland; and Seppo Horsmanheimo at VTT.

For the financial support I have received during my time as a PhD student, whether directly or indirectly, I thank the following: the Academy of Finland projects FResCo, EFFIMA, eDiHy and DIJON; the European Commission project DEPLOY; the former Department of Information Technologies and current Computer Science subject at Åbo Akademi; TUCS – Turku Centre

for Computer Science; Stiftelsen för Åbo Akademi; Svensk-Österbottniska Samfundet r.f.; and the Rector of Åbo Akademi.

I would also like to thank the people I have spent time with outside work, including the photography association Fotoklubben Pictura vid Åbo Akademi r.f., the folk dancing association Folkdanslaget Otakt vid Åbo Akademi r.f., and the people with self-proclaimed strange interests in Föreningen för Underliga Intressen vid Åbo Akademi r.f., as well as my other friends, both from long ago and recent ones. My thanks also to my parents, Märta and Jean; while they have not been directly involved in my work they have encouraged me to seek my own path and helped me in so many other ways. Also thanks to my grandparents; although my grandfathers are no longer with us, I am sure that they would have been pleased that I have made it this far; and to my brother Simon, his girlfriend Jeanette, their children, and the rest of my extended family.

Finally, a very special thank you to Diana Gratie, for everything.

Petter Sandvik
Åbo, October 2015

List of Original Publications

I Petter Sandvik and Mats Neovius. A Further Look at the Distance-Availability Weighted Piece Selection Method: A BitTorrent Piece Selection Method for On-Demand Media Streaming. *International Journal on Advances in Networks and Services*, ISSN 1942-2644, Vol. 3, No. 3 & 4, pp. 473–483. IARIA, 2010
`http://www.iariajournals.org/networks_and_services/`

II Petter Sandvik and Kaisa Sere. Formal Analysis and Verification of Peer-to-Peer Node Behaviour. In: Antonio Liotta, Nikos Antonopoulos, Giuseppe Di Fatta, Takahiro Hara and Quang Hieu Vu (Eds.), *The Third International Conference on Advances in P2P Systems (AP2PS 2011)*, pp. 47–52. IARIA, 2011.

III Luigia Petre, Petter Sandvik, and Kaisa Sere. Node Coordination in Peer-to-Peer Networks. In: Marjan Sirjani (Ed.), *COORDINATION 2012*, Lecture Notes in Computer Science Vol. 7274, pp. 196–211. Springer-Verlag GmbH Berlin Heidelberg, 2012.

IV Kristian Lumme, Luigia Petre, Petter Sandvik, and Kaisa Sere. A Formal Approach to H.264 Video Decoding on Multicore Systems. *Int. J. Critical Computer-Based Systems*, Vol. 4, No. 1, pp. 3–26. Inderscience Publishers, 2013.
`http://dx.doi.org/10.1504/IJCCBS.2013.053740`

V Petter Sandvik. Translation and Validation of SPECTA – A Specification Language for Content Transfer Algorithms. 2015.

Contents

Part I

Research Summary

1 | Introduction

As far back as we know, people have always had a desire to preserve and share their memories. Interpretative ways of doing this, such as remembering and telling stories, or creating paintings or sculptures, are well known from history. However, these memories were always processed and filtered by human beings, and although technological advances such as the printing press made these representations easier to share, they still remained human interpretations of a phenomenon. It was not until the 19th century CE that, thanks to technological progress, a memory in the form of sound or vision could be preserved without necessarily being reinterpreted by a human being. Having a needle etch a groove into a roll of wax could preserve sounds, and light hitting a chemically treated silver plate could make a visual snapshot. Timed sequences of photographs became a way of storing moving images. These technological breakthroughs can be seen as setting the initial stage for media distribution.

Common for all these early ways of preserving audiovisual memories is that they preserved memories as physical representations. The groove etched into a sound recording was, hopefully, a miniaturised physical representation of the sound waves that would have reached a listener's ear at the time of recording. The light reflected from a photographic image was intended to be the same as the light that would have reached an observer's eyes when the photo was taken, if the observer had been where the photographic equipment was. Over time the technology moved on from purely mechanical recordings to other types, such as the magnetic representation used in tape-based audio and video. Even today, storing audiovisual data as an analogue representation on a physical object is still around. As recently as in the second half of the 20th century this was a prevalent way of storing audiovisual memories, as well as sharing these by sharing the physical objects, in the form of vinyl records, photographic film, and tape-based media formats.

One problem with these analogue, physical recordings was their fragility. They could be easily damaged, often even just by playing them back, but at the same time they could not be easily copied without a degradation in quality. This made long-term preservation of the recorded content difficult. Fortunately, the arrival of computer technology provided a solution. Our

computers are usually binary in the ways their signals are represented, but in the very least they are digital. Therefore, they use only discrete signals, and for them to store data the representation thereof must be abstracted away from the real world media it is stored on. In the simplest form, this means that the physical object used to store data may be damaged, but unless the damage is large enough that it causes the computer to treat a number as another number, the digital data is perfectly preserved. Moreover, this digital content data can be amended with additional data for error correction, by which even larger damage can be recovered from without any loss of actual content. To the general public, this concept was introduced when the compact disc (CD) was released to the public in 1982 [110]. Onto this small silver-coloured disc stereophonic sound was recorded as a digital interpretation. The signal was read from the disc by means of a laser passing through a transparent plastic layer on one side of the disc but reflected off a thin metal layer inside, and although the initial claims of the indestructibility of the discs were exaggerated, they were much more tolerant of scratches and dust than their analogue predecessors. This was the start of the second stage of media distribution, in which the content was in a digital format, but the content distribution was still physical.

After the compact disc, other physical, but digital formats followed, such as digital audio and video tapes and the digital video disc (DVD). Although the data on a compact disc was digitally encoded, it was not *compressed*, unlike these newer formats. When digital data is concerned, compressed means that mathematical operations have been performed on the data, allowing more data to fit onto the same logical or physical space. If this has been done in such a way that the original data can be fully restored, the compression is known as lossless. However, for audiovisual data, lossy compression is often used, in which not only completely redundant data is removed to save space but also those parts of the signal that are the least perceptible to human senses. As computing power increased, these mathematical algorithms could be more advanced and therefore efficient in minimising the space used by the content. Towards the end of the last decade of the 20th century these advances, along with other technical improvements, had reached the point where it became feasible to share the compressed media content directly between computers by means of a network, instead of having to store them on a removable, physical object.

Thus, the evolution of media distribution can be seen to have three different stages, shown in Figure 1.1. In our work we have focused on the third of these stages, digital media distribution, where content is in a digital format and distributed digitally. Digital media has become very important for spreading news and information [62], as well as entertainment, and the impact of improvements to the technologies used herein would therefore be seen even in fields outside of computing. Many digital media distribution systems are

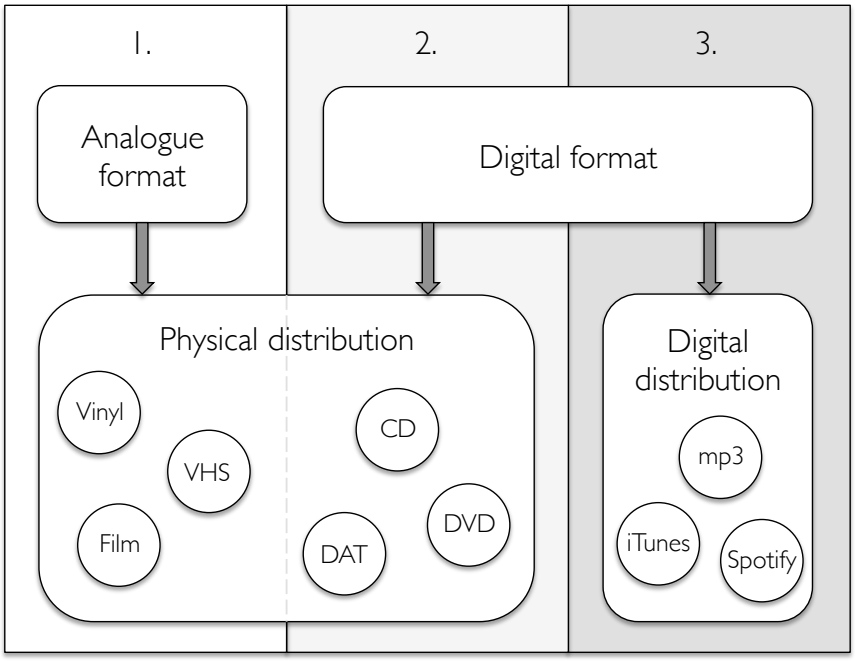


Figure 1.1: The three evolution stages of media distribution

already in use, some of which we take a look at in Chapter 3, and their usage seems to be increasing [54, 55]. While this has highlighted both the need and the interest for this type of media distribution, the practical applications seem to be fraught with problems, diminishing the quality of the experience. A survey in 2015 found that only 1 in 4 people persevered more than 4 minutes when faced with a "sub-par experience" for viewing on-demand streaming video, such as "poor picture fidelity and excessive stream interruptions" [61].

Overall, the subject of how technical problems with digital media distribution affects the user experience has been studied for some time [58, 101]. However, the user experience is a whole domain in itself and not the focus of our work. We instead note that it appears as though digital media distribution still faces a few challenges before it can properly serve its purpose. One of these challenges concerns the performance; as media distribution systems scale up, the act of getting the content to the consumers may require more resources than the systems were originally designed to handle. Furthermore, the content may be transferred over networks that were not initially designed for the type, or amount, of data involved. This will become more prevalent as content quality and sizes increase. Another challenge is that limitations in bandwidth and processing power impose restrictions on different parts of the distribution process. This can be seen particularly in developing countries,

where the infrastructure may be lacking, but also in the industrialised world, where new use cases may challenge the existing methods.

In addition to the existing challenges we note a few requirements, or at least desirable properties, of a digital media distribution system. To meet current and upcoming demands, the system should be efficient enough to handle big data without sacrificing performance. Security and reliability are also a cornerstone, making it possible for the system to be trusted by the content consumers as well as the producers. Our aim is to showcase how these properties can be satisfied and improved upon through formally proving correctness, specifically with regards to the software algorithms involved in the process of digitally distributing media content.

We begin in Chapter 2 by describing what digital media distribution entails, and we continue in Chapter 3 by describing existing media distribution systems. Our work on improving media distribution is based on formal methods, which we describe in Chapter 4. In Chapter 5 we discuss how to use the formal modelling techniques in order to improve digital media distribution, based on our previously published work. In Chapter 6 we discuss related work. We conclude Part I of this dissertation in Chapter 7 with discussion and future work. After that, Part II follows with reprints of our previously published articles.

2 | Digital Media Distribution

We start by defining what we mean by the notion of *digital media distribution*. In our case, the term *media* refers to any audiovisual content, which includes sound and (moving) pictures. While digital media could also include text-based formats such as e-books, as well as content consisting of a combination of different content types, we have not taken these into special account. As mentioned in Chapter 1, *digital distribution* is the act of distributing content over a (digital) computer network or series of computer networks, such as the Internet. Digital distribution is opposed to *physical distribution*, in which the content is distributed by recording it onto physical objects and then physically moving those objects. We note that digital distribution requires the content to be in digital format, unlike physical distribution, where content may be in either digital or analogue format.

At this stage it is appropriate to point out that there exists another type of media distribution method, which is not physical in the sense of transporting physical objects with content on them. This media distribution method is the traditional *broadcast*, in which the same content is transmitted simultaneously from one sender to many receivers using radio waves or cables as the distribution medium. The content of this type of broadcast can be in analogue or digital format, and in the latter case the same content is also often transmitted by other means, such as over the Internet. In our work, we have not specifically considered broadcasts, as the one-sided and simultaneous nature of broadcasts makes them less interesting from the point of view of media distribution research. However, the worldwide trend is to move away from analogue broadcasts towards digital, and these can be seen as a part of digital media distribution in general. Thus, while we do not specifically consider broadcasts in our work, many of the points discussed further on can be applied to digital broadcasts as well.

2.1 The Digital Media Distribution Process

If we think of the digital media distribution process as the whole chain of actions involved in getting a recording of a physical, real-world phenomenon

and then transferring it in digital format to some other place in order to be reproduced at a later time, it can be seen as containing the following simplified steps, as shown in Figure 2.1:

1. **Converting the real-world phenomenon to a digital representation.** For instance, images are typically captured by digital image sensors and converted into bits by the associated processing units, while audio can be captured by microphones and converted by analogue-to-digital converters into sequences of bits.

2. **Encoding the digital representation for distribution.** For instance, a series of still images can be combined into a motion picture, as well as compressed and encoded into H.264 video format.

3. **Transferring the encoded content over a network.** This includes setting up and routing data within overlay networks, and algorithms for efficient data transfer.

4. **Decoding the encoded content.** Reconstructing uncompressed, independent data from the transmitted format containing encoded and packed data.

5. **Converting the decoded digital content back into a real-world phenomenon.** This includes various types of displays for showing images and digital-to-analogue converter and loudspeaker combinations for reproducing audio.

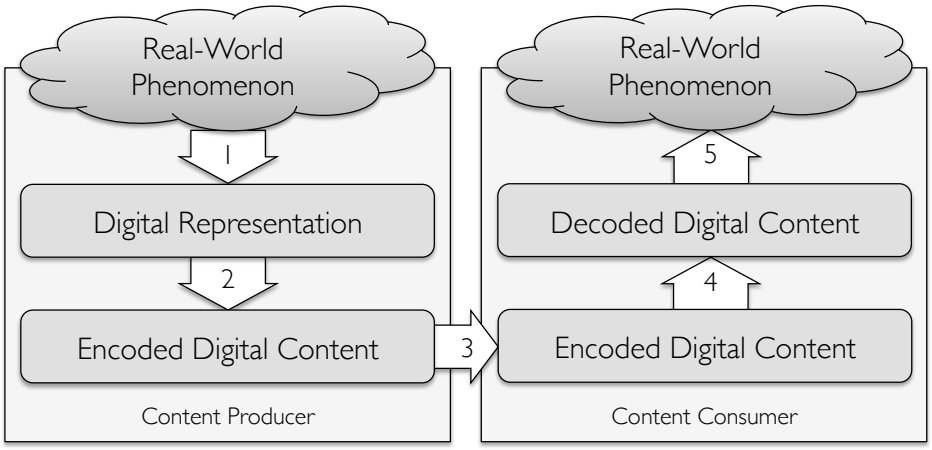


Figure 2.1: The media distribution process, consisting of converting a real-world phenomenon into digital content at the Content Producer side, transferring it, and converting it back at the Content Consumer side.

We note that between any of the above steps, i.e., as soon as the data is in a digital format, it is technically possible to store an exact copy of the intermediate data. In practice, storing the raw, uncompressed data can be impractical, and for instance many digital video cameras store data already compressed and encoded. Therefore, step 2 can often be executed more than once. The re-encoding from one format to another is known as *transcoding*, and can change not only the encoding format but also other properties such as bit rate [79] and resolution [138], and is often done to prepare a variety of final distribution versions from one master copy. For quality purposes it is preferable to do editing and combining of content before any lossy encoding has been done, although this may not always be possible in practice. Also worth noting is that in the case of computer-generated images (CGI) or audio, the content is already from the beginning in a digital format, and therefore in such a case the initial step in the above list is superfluous.

When it comes to our work, not all of the steps above are relevant to include. Steps 1 and 5, converting from analogue to digital and vice versa, are usually handled by hardware to an extent that leaves them out of scope for our work. We also note that while the same content is produced only once it can be consumed many times, and likewise, it could be encoded only once in any specific format, but distributed and decoded many times. For that reason, we do not focus our work on step 2, i.e. encoding the media, except that in Sections 3.4 and 6.3 we discuss related work within this field. Instead, we focus on the parts of the process where the content consumer is involved, including steps 3 and 4 above – **transferring and decoding the media**. These are also steps in which correctness is a central part; the content should be identical after it has been transferred, and a specific content should be decoded in a certain way. In contrast, there is no single correct way of getting a digital representation of a real-world phenomenon, or for that matter getting encoded content from the "raw" digital content.

In the following, we look at a few existing techniques for media distribution networks, starting with the transfer process and continuing onto media decoding as well as systems that integrate both.

3 | Media Distribution Systems

Digital media distribution systems have many parts, but the main ones we focus on are getting content across the network to the consumer, and decoding the content such that it can be consumed. We begin with the former in Section 3.1, and the latter we describe in Section 3.4. We continue in Section 3.5 by describing a few existing systems for digital media distribution.

3.1 Media Transfer Methods

For transferring digital media over a network, there are two general methods that we define in the following. While *downloading* can refer to any transferring of data to a client requested by the client, *streaming* is 'play-while-downloading' rather than 'open-after-downloading' [148]. Thus, in our work *downloading* refers to when content is transferred to the initiator of the transfer for the purpose of being used (e.g., played back) at a later date, after it has been transferred in its entirety. In contrast, *streaming* refers to when content is transferred for the purpose of immediate use (e.g., playback), even before the entire content has been transferred. Streaming can be further separated into *live streaming* and *on-demand streaming*. The former is similar to a broadcast in that every participating client receives the same content and plays it back at approximately the same time, and the entire content may not be available when transmission begins. As a contrast, with on-demand streaming the content already exists, and individual clients can choose to request the content to be transmitted to them for playback.

The separation between downloading and streaming implies that downloaded content is meant for possible long-term client-side storage, while streamed content is only meant to be stored temporarily by the computer accessing the content. This difference may seem trivial, and in many cases it is more of a legal or licensing issue rather than a technical one, but in one area this distinction is important even in a technical sense: the transfer order. A lot of content delivered over the Internet is transferred in-order,

i.e., the order in which the content is transferred is the same as when the content is stored. For instance, web content is delivered via the hypertext transfer protocol (HTTP), which was designed for in-order transfers. However, for large content, including digital media, transferring content out-of-order can be preferable for improving performance, reliability and availability of content. Because playback of content happens in-order, the out-of-order transfer algorithms sometimes used for downloading content cannot be used without modification for content that should be played back while downloading, i.e., streaming content. We discuss more about out-of-order transfers in Sections 3.3.1 and 5.2.

For both downloading and streaming there are existing solutions in use for media distribution. Some of these are more general-purpose, or *open*, and others are proprietary or specific to a certain company, i.e., *closed*. Of the general-purpose solutions we focus on two: content delivery networks in Section 3.2 and peer-to-peer networks in Section 3.3.

3.2 Content Delivery Networks

A content delivery network (CDN) consists of a network of servers, distributed over a wide area and replicating, at least partially, the same content. CDNs have existed since the late 1990s, and the idea behind CDNs is to have content located geographically, topologically or latency-wise closer [85] to clients, usually in order to improve response times or reduce load on the original server. Content providers contract with CDN providers to host and distribute static content, while the content providers can host dynamic content such as changing web pages themselves, embedding the static content if necessary. Exactly how the CDNs are integrated into the content distribution depends on the solution chosen, with the CDN servers replicating anything from entire servers to a subset of the content to be distributed [75, 85, 124].

While CDNs are used by the majority of the largest web sites [35], and sites such as YouTube and Facebook rely heavily on their own CDNs, a few things have limited their universal adaptation in media distribution. Firstly, it can be difficult to integrate CDNs with different access policies, such as when requiring a user to log in in order to access a particular content. Secondly, while web pages usually include many small files requested separately and therefore benefit greatly from the the lower access latencies provided by CDNs, media distribution handles mostly fewer, larger files, which are more often limited by transfer rates than latencies. In Section 3.5 we will take a look at integrated media distribution systems, which in many cases employ CDNs, but in the following we will look at a different technique also used for media distribution.

3.3 Peer-to-Peer Networks

Schollmeier [125] defines *peer-to-peer* as follows:

> A distributed network architecture may be called a Peer-to-Peer (P-to-P, P2P, ...) network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers, ...). These shared resources are necessary to provide the Service and content offered by the network (e.g. file sharing or shared workspaces for collaboration). They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource (Service and content) providers as well as resource (Service and content) requestors (Servent-concept).

In general, a peer-to-peer network is a network in which the clients may also act as servers, and therefore communicate with and send data directly to other clients, or peers. Furthermore, a major difference between a CDN and a peer-to-peer network is that while a CDN is an organised, usually homogenous network made up of nodes controlled by, or on behalf of, the content producers, a peer-to-peer network is often heterogenous, less organised, and made up of nodes controlled mainly by individual consumers of the content.

Peer-to-peer networks can be classified according to their structure. On one end of the scale we find "pure" peer-to-peer networks, which are completely decentralised and in which all peers have equal status. Further towards a centralised structure lie mediated peer-to-peer networks, in which a central server may be responsible for control of the network while data transfer may happen directly between peers, and hybrid peer-to-peer networks, in which peers have an internal hierarchy depending on, for instance, bandwidth or processing power [14].

While peer-to-peer network architectures have many uses, including voice-over-IP (VoIP) [17, 80] and instant messaging protocols [63, 155], peer-to-peer networks rose to prominence during the 1990s as part of file sharing applications. Before server virtualisation became simple and commonplace, the act of setting up a server was costly and cumbersome. Thus, the ability to share files directly without a dedicated server was welcomed by users, especially combined with the ability to find new content. As sometimes happens with technology, the fastest adoption does not necessarily take place within the limits of existing laws, and thus many of the early peer-to-peer file sharing applications became infamous because of their perceived link to unauthorised content distribution. These include Kazaa, which before its demise as a peer-to-peer file sharing application used the FastTrack protocol and therefore a hybrid peer-to-peer network [90], and Gnutella [114], which

is a "pure" peer-to-peer network file sharing protocol used by the infamous LimeWire application [92]. More thorough comparisons of these have been done [9, 118], but here we note that while most peer-to-peer file sharing applications aim to quickly locate content [67], one whose aim is to quickly transfer content is BitTorrent [40]. This makes BitTorrent interesting from a media distribution perspective, and as it has kept its popularity among peer-to-peer file sharing [54, 55] and is important to our work, we will in the following describe BitTorrent in more detail.

3.3.1 BitTorrent

BitTorrent is a file sharing application introduced in 2001 [39]. While the original BitTorrent protocol [40] resulted in a mediated peer-to-peer network type, later additions have evolved the structure towards a combination of different types. Unlike many other peer-to-peer networks, there is no single network connecting all users of BitTorrent, but rather a separate subnetwork for each set of content, or *torrent*, uniquely identified by its *info hash*. In order for a peer to find other peers, services called *trackers* keep track of peers involved in sharing a particular torrent. Furthermore, each torrent is split into *pieces*, of typically 256 kB or more in size, each of which is further split into *blocks* of 16 kB [89]. A peer will keep track of which pieces are held by the other peers it is connected to, and request blocks of a piece from peers who have the piece that those blocks belong to. After a peer has received all the blocks in a piece it may offer that piece for distribution to other peers.

One of the key ideas to make this system work is that a peer should not request the pieces in-order, but rather randomly until one complete piece has been transferred. After that BitTorrent switches to using the *rarest-first method* [40]. This means that the piece with the lowest availability, i.e., the piece that the fewest active peers have, should be requested before pieces with higher availability. The behaviour when more than one piece has the same availability is not defined [40]. By requesting pieces rarest-first the algorithm tries to ensure a reasonably even distribution of pieces, compared to a situation where all peers would start by initially requesting the same pieces. This is also a reason as to why new peers start with requesting pieces randomly, as peers joining soon after each other would otherwise all start by requesting the same piece.

To create an incentive for peers to upload as well as download data, the reference implementation of BitTorrent uses a *tit-for-tat* mechanism. Each peer evaluates which of the other peers have sent data to it, and those peers who have sent the most data, plus one random peer, are allowed to receive data in return [40, 89]. Other implementations that used different methods of peer selection have been proposed, including BitTyrant [109], which carefully estimates which peers to send data to in order to maximise its

own download speed while minimising the amount of data sent to other peers. In any case, the tit-for-tat mechanism only works when a peer is uploading and downloading simultaneously, and therefore clients switch to other peer selection methods when *seeding*, i.e., when they have a complete copy of the torrent data but still participate in sharing the content by sending data to other peers. Originally the peers who could receive data fastest were chosen for sending data to when seeding [40], but as this could lead to unfairness, later clients switched to choosing these peers randomly [109].

Originally there was no built-in content search in BitTorrent, and instead search for content was carried out using web search pages. These web pages allowed users to download so-called *torrent index files*, containing metadata about each torrent, including its info hash, piece size, names and sizes of included files, URL(s) of tracker(s), and SHA-1 hashes of all the pieces. However, BitTorrent later evolved to include other methods of discovering peers beside the tracker. These include finding them using a distributed hash table (DHT), where lookups are performed using an overlay network, and peer exchange, in which peers directly exchange information about which other peers they have experience of being involved in sharing the content. Thus, so-called *trackerless* torrents were introduced, and taking the decentralisation one step further was the use of *magnet links* [86]. A magnet link contains the info hash, with which the DHT can be used to retrieve the torrent metadata from other peers, thus bypassing not only the tracker but also the retrieval of the torrent index file from a web page.

The BitTorrent ecosystem is, at the time of writing, very much alive and thriving for content distribution, both authorised and non-authorised. In a snapshot of the second half of 2014 BitTorrent comprised approximately 25, 36, and 56 percent of upstream Internet traffic during peak periods in North America, Europe, and the Asia-Pacific regions, respectively [55]. Besides its infamous use in unauthorised distribution of music, video, and software, it has also been used for distributing content updates in games such as World of Warcraft [77, 91] and for distributing code to servers used by Facebook [41, 49] and Twitter [82]. BitTorrent has also been reported as being used by YouTube for internal transfers inside and between data centres, with YouTube being "very pleased with the performance" [22]. However, these usages are all of the download-type, in which the content must be transferred in its entirety before it is certain that it can be used. As previously mentioned, the out-of-order nature of BitTorrent that makes it efficient at transferring large data sets is not inherently compatible with the in-order nature of streaming content. There have been many proposals for adapting BitTorrent to streaming [32, 33, 103, 120, 129, 145], but in practice streaming of BitTorrent content seems to be done by naïvely requesting the pieces in-order [23, 57], despite the well known shortcomings of using this method in a BitTorrent network [106]. We return to this issue in Section 5.2.1.

BitTorrent is by its very nature an open and unregulated system, and although there is a company ostensibly in charge of BitTorrent (or at least the "official" client software), many of the features have been introduced by individual developers and in response to different user needs. This lack of control over BitTorrent, along with its reputation as a method of choice for unlicensed content distribution, have probably contributed to the overall lack of support from large content producers. In Section 3.5, we look at a few different media distribution systems that have seen more mainstream use. But before that, we turn our attention towards how to decode the received media for playback.

3.4 Media Decoding

As mentioned earlier, content that is transferred over a network is usually encoded in such a way that it is compressed, reducing the amount of data. With audiovisual data, the most common way of doing this is by discarding the least important information. This way of reducing the amount of data has its root in a time before digitalisation, when it often arose out of necessity; for instance, analog colour video signals for TVs separated the luminance and colour information and put a greater emphasis on the luminance signal, not only because the human eye is more sensitive to luminance, but also because compatibility was needed with the previous black and white TVs. The separation of luminance and colour is still prevalent in digital video, and for lowering the data rate without affecting the perceived quality, it is common for each of the two colour channels to contain only a quarter of the detail of the luminance channel [45]. Similarly, when sound recordings on vinyl records moved from single channel (mono) sound to dual channel (stereo), the grooves in the vinyl did not encode left and right channels separately but rather their combination ($L + R$) in a horizontal groove and their difference ($L - R$) vertically [59], which not only allowed greater fidelity for the combined channel but also provided compatibility with mono equipment. This idea has also survived the digital transition; for instance, the joint stereo encoding used in MPEG-1 Layer III (MP3) and MPEG-2/4 Advanced Audio Coding (AAC) [25] formats can similarly use mid/side-stereo encoding [52]. In this encoding the combination of the left and right channels is encoded as one channel and the difference as another [68], thus allowing more data for the former when there is not much difference between left and right channel audio.

When it comes to encoding video content, in practice a large part of the content is such that many parts of the image remain static for some time, or at least do not change very much. In video encoding it is therefore common to take advantage of the similarities between one image, or frame, and the next.

This means that instead of encoding each frame of the video independently, the encoder introduces dependencies between frames. We explore this further in Section 5.3.

While the aforementioned techniques reduce the data rate for encoded media somewhat, a major reduction in data rate comes from the use of lossy encoding mentioned in Chapter 1. Lossy encoding means that the mathematical transformation of the content does not perfectly preserve the content, but discards some of the data. This should be done in such a way that both the perceived loss in quality and the amount of data needed to represent the content is as small as possible. Thus, there are many ways of creating lossy compressed media from uncompressed. Encoding is quite a complex task, and for some parts a "brute force" method can be employed; the encoder tries different strategies and in the end chooses the one that appears to give the best result for a given piece of content. Such a method is trivial to parallelise, and the video encoding process has long been an area that has taken advantage of parallel processing to improve speed and efficiency, as evidenced by the work of for instance Li et al. [93], Xu et al. [150], and Yung and Chu [152]. However, because a particular content is typically decoded many more times than it is encoded, the cumulative possible gains in efficiency are much larger for decoding than encoding. Thus, in our work we focus on the decoding process.

There are many different audiovisual formats that media can be encoded in. However, the formats in common and contemporary use are not very different from each other in their approach. For audio, most coding standards are based on psychoacoustic modelling [137, 156] to choose how to discard the sounds that are the least perceptible to human ears, thereby reducing the data rate. Also, many formats use a Huffman-style lossless encoding of the loosely encoded content to further reduce the data rate [137, 156]. For distribution of solely audio, MP3 and AAC [25] are common formats, with Ogg Vorbis [100] as a "free", or supposedly less patent- and licensing-encumbered, option. For stereo audio, there is little practical quality difference between these formats at the bit rates commonly used for audio distribution. Audio included with video commonly uses AAC or Dolby Digital (AC3) audio, the latter being carried over from its use for surround audio in DVD and Blu-ray digital video disc formats. For the video content itself, H.264 [66] is a common format at the time of writing, but there are others, such as VC-1 [69] and VP8 [16], that support similar features. These features include a 16x16 pixel block size, integer DCT transform, and deblocking filters [37, 69].

Regardless of the formats used, the content has to be distributed somehow. As mentioned previously, there are existing systems for media distribution, and in the following we take a look at a few popular and successful ones.

3.5 Integrated Systems

A variety of different existing media distribution systems are in use, and many of these are *integrated*. By this we refer to a media distribution system where the content, distribution mechanism and, to some extent, playback software are all controlled by the same entity. Most of these tend to focus on streaming rather than downloading, presumably because the immediacy of streaming makes monetisation (through the use of advertisements or subscription fees) easier than in a situation where the content consumption is carried out at a later date. In the following, we shortly look at a few of the more popular and interesting integrated media distribution systems.

3.5.1 YouTube

YouTube is an advertising-supported video sharing web site, which launched in early 2005 [147] and was bought by Google the following year [126], subsequently becoming Google's premier video sharing service. Originally videos were encoded with the Sorenson Spark H.263 codec at a resolution of 320 by 240 pixels, with audio in 64 kbps MP3 format, and displayed in web browsers using the Flash plug-in from Macromedia (later from Adobe). The range of different encodings has since grown to include resolutions up to Ultra HD (also known as 4K or 2160p) as well as different codecs corresponding to different standards and for compatibility with different devices, and playback methods. Since late 2013, Dynamic Adaptive Streaming over HTTP (DASH) [130, 133] has been required for video resolutions above 720p. Using DASH, the same content is encoded with different resolutions and bit rates and split into sequences of small segments. The playback application must then request individual audio and video segments, which may be of different qualities to suit changing conditions, and combine these segments client-side [130].

In a snapshot from the second half of 2014, YouTube was the largest or second largest user of Internet bandwidth in all examined regions of the world except Africa [55]. YouTube used to function using third-party CDNs, which was very expensive [131], and YouTube has therefore built a CDN of their own, which has been extensively studied [113, 143].

3.5.2 Netflix

Netflix is the leading provider of a subscription service for online movies and TV shows [6]. In a snapshot from the second half of 2014, Netflix was reported to comprise 34.9% of downstream fixed line Internet traffic in North America during peak periods [55], and earlier that year it was reported to comprise 17.8% of downstream traffic in the British Isles during "peak evening hours" [54]. As Netflix is only available in the Americas and in a limited

number of countries in other parts of the world, mainly in Northern Europe, its impact on the global Internet traffic overall is less severe.

In the United States, Netflix began as a mail rental service for DVDs in 1997, and after expanding to digital distribution in 2007, by 2010 on-demand streaming had turned into "a major part of Netflix's business" [126]. In other countries, Netflix offers on-demand streaming only. Initially, this on-demand streaming of media was served by Netflix's own data centres, but then expanded to use CDNs such as Amazon, Limelight and Akamai [6]. More recently, Netflix begun placing its own "Open Connect" CDN servers directly at different Internet service providers, thereby letting the traffic flow only locally from the ISP to the end users, reducing the amount of Internet traffic used by the Netflix streaming service. Outside the US, all Netflix traffic now goes through Netflix's own CDN [27]. This move, while beneficial to the ISP and end users alike, has met some resistance from many Internet service providers [87], including those who are also offering cable TV and therefore view Netflix as a competitor in media distribution. Netflix has also been interested in augmenting its CDN with a peer-to-peer network [26].

3.5.3 Spotify

Spotify is an on-demand music streaming service, initially launched in October 2008 in a few European countries [56, 95]. The company, Spotify AB, has developed its own, proprietary applications for both mobile and desktop operating systems [56], as well as a browser-based solution. While the basic service is ad-supported and does not require payment from a user, additional features – such as offline listening, higher audio bit rate, and ability to connect different devices – requires a monthly fee [84], and the exact differentiation between paying and non-paying users has changed over time.

Originally Spotify used a peer-to-peer network as well as their own servers to deliver music content [84]. Each desktop client connected to Spotify would function as a peer in the peer-to-peer network, caching a certain amount of recently played content. When streaming a music track not in the peer's own cache, the client software would start by getting the first piece of the content from Spotify's servers, and then use the peer-to-peer network to find other sources for the requested content. For this the software would use techniques inspired by peer-to-peer file sharing, both "Gnutella-style" and "BitTorrent-style" [83, 144]. The former means that a peer asks its neighbours and neighbours' neighbours for possible sources for the content, and the latter means asking a central tracker for other peers interested in the same content [56]. However, instead of a random subset of peers as in standard BitTorrent the tracker would return the most recent peers, because of the greater possibility that a peer that has recently requested the content would still hold the content in its cache. The actual content transfer would happen

in-order, as the data amount was very small compared to video content [83], and therefore it was assumed that any gains from the use of out-of-order transfer would not be worth the additional complexity.

Mobile clients for Spotify have not participated in the peer-to-peer aspects of Spotify [83, 144], and the use of mobile clients has increased over time. Spotify has therefore decided to "phase out" the use of peer-to-peer technology, because they believe that they can now serve users better using their own network of servers [48].

3.6 The Research Question

Looking at the existing systems, we are interested in finding out how we can improve the performance and efficiency of digital media distribution. We have noted the following three main areas of possible improvement:

1. **Network structure and coordination.** Media distribution systems seem to alternate between first-party and third-party solutions, the former being more cost-effective and easier to control while the latter allow for more rapid expansion and larger reach. Regardless of which type the focus is on at the moment, there is a clear need for methods of coordinating and controlling an expanding network used for distribution.

2. **Transport of content over the network.** There is a need for solving the issue of getting content across the network as efficiently as possible; not only to the consumers, but also to other points in the distribution network.

3. **Content encoding.** There is a need for having the content in a format that uses as little data as possible, for easier transport over the network. However, the encoding used should allow for efficient decoding in order to get the content onto such a wide variety of devices as possible.

For all of the above, reliability is an important goal, because we want the content producers and consumers alike to trust our media distribution system. For this, we need the network itself to be reliable, we need the transport of content over the network to be reliable, and we also need the decoding of content at the consumers' side to be reliable. In a distributed network, there are ways of improving the overall reliability of the system while the individual parts are unreliable [1]. This is indeed a useful feature, for instance when dealing with changing conditions outside of our control, but our issue with reliability requires more than this. Ideally, we would like to be certain that the software running on each node of our media distribution network behaves correctly and without errors, and that we can showcase efficiency without compromising on the correctness. While it could be argued that

this is, or at least should be, the goal for any software, in most cases the methods used in the development do not even strive to be able to guarantee correctness. It would be a good idea to approach this systematically, with correctness as a cornerstone. For this we need a specific approach to software development, and in the following chapter we will describe such an approach: formal methods.

4 | Formal Methods

When a physical system of large scale – such as a car, a bridge, or a building – is to be constructed, there is often a model of it created first before the design is finalised and the actual construction can begin. Traditionally, this model has been a physical object different from the intended final product in some ways, such as in scale, structure, or the materials used. However, for such a model to be useful it needs some properties similar enough to those of the intended final product; the model can then be used to evaluate some aspects of the design before it is finalised. These days, such models are constructed with computers and may never enter the physical world, but they are still used for creating technical drawings that specify these systems and can be used to reason about them before they are built. Such a technical drawing is often known as a *blueprint* [3].

The construction of software systems has traditionally taken a different approach. In the development of software it is inevitable that into the code there will at some point be introduced faults or flaws, colloquially known as bugs. Because this is undesirable, the bugs should be found so that they can be corrected, which in many cases is done using testing. As pointed out by Dijkstra, testing "shows the presence, not the absence of bugs" [29]. One way of dealing with the problem of bugs can be formal verification, in which the code is checked to ensure mathematical correctness before it is deployed. For instance, such a system is used for the mobile applications by Facebook [30]. While this type of verification improves the quality of the system, it does not automatically lead to a better understanding of the system. We could instead reason about the problem the other way around, by first thinking about what the system should achieve and then making sure that what we end up creating does not contradict that. This is known as correctness by construction [88].

Stepwise development can help to ensure correctness. Most approaches to software development, including the traditional waterfall model [20] as well as the newer agile development [18] methods, follow a stepwise model, with these steps referred to as life cycle stages in the waterfall model and iterations in the agile methods [64]. However, while this stepwise development is intended to improve the quality of the software, it is not done in a way that guarantees

correctness and desirable properties. If we instead employ strict mathematical-logical ways of reasoning about correctness, we can *prove* correctness and desirable properties, under certain assumptions. Such methods of enabling correctness by construction are known as *formal methods*, and they are used to build *formal models*, which are "blueprints adapted to our discipline" [3].

In the following, we will describe the formal method we have used for our work, as well as tools and add-ons supporting this method.

4.1 Event-B

The B-Method [2] is a formal method for specification and development of highly dependable software, and it has been used in several large industrial projects [15, 19, 53]. Event-B [3] combines the B-Method with ideas from the Action Systems [10, 12, 146] framework, to create a way of modelling reactive, event-driven systems.

Event-B is a state-based formal method. This means that the state of the system is modelled using *variables*, and transitions between these states are modelled using *events*, which contain *actions* that update variables. For a particular transition to happen, certain preconditions must exist, modelled as event *guards*. The guards consist of predicates that must evaluate to TRUE whenever the event is *enabled*, i.e., when it is possible and desirable for the event to occur. The variables and events in an Event-B model of a system are contained in a *machine*, also referred to as the "dynamic part". An Event-B machine may *see* one or more *contexts*, known as the "static part", containing constants and (carrier) sets as well as axioms about these. Fundamental to an Event-B machine are *invariants*, which state properties in the machine that must hold for every reachable state of the model, i.e., every possible combination of values that the variables could have obtained based on the events. Initially the invariants must be satisfied by the values given to variables in the special INITIALISATION event, and none of the other events in the machine may change variables in such a way that the invariants are contradicted. The general structure of an Event-B machine and context can be seen in Figure 4.1.

A key concept in formal modelling with Event-B is *refinement*, which is the gradual introduction of new events, variables or data, making the system progress stepwise from its original specification towards a more concrete implementation. In other words, we start from a model of what the system should achieve and work stepwise, mathematically proving correctness along the way, towards a model of how the system should achieve its intended purpose. Horizontal refinement, also known as *superposition refinement* [13, 76], is the term used when we refine a model by adding new variables and events to the existing model. To ensure provable correctness, these newly

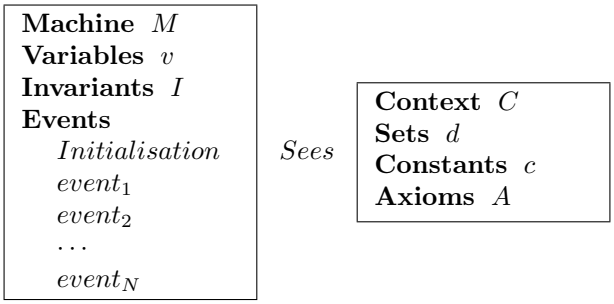


Figure 4.1: A machine M and a context C in Event-B, based on Abrial [3].

added variables and events must not contradict any previous, more abstract model. Vertical refinement, or *data refinement* [11], is the term used when we replace an abstract representation of data with a more concrete one, and change the events accordingly. In this case, we need to add *gluing invariants*, which formally define the relationship between the previous, abstract variables and the newly introduced, concrete ones.

While machines and contexts are useful for structuring a model, the real reason for modelling using Event-B is the previously mentioned ability to prove correctness and desirable properties. What is to be proved for an Event-B model is defined by *proof obligations*. These are sequents or mathematical formulas that can be used to show that the actions of the events do not invalidate any assumptions given about the variables. While these can be generated by a human being, doing so would be "foolish (and error prone)" [3] due to the large number of proof obligations that often can be needed. Furthermore, not only generating the proof obligations but also *discharging* them, i.e., *proving*, is something that we should preferably be able to do automatically. For this to happen, we need tool support.

4.2 The Rodin Platform

The EU project Rodin (Rigorous Open Development Environment for Complex Systems) [115] ran from 2004 to 2007, and during it and its follow-up project Deploy [44] the Rodin Platform tool [4, 5, 50] was developed. The Rodin Platform tool is based on the Eclipse platform, an "open source, robust, full-featured, commercial-quality, industry platform for the development of highly integrated tools and rich client applications" [47].

The Rodin Platform tool helps with both modelling and proving with Event-B. Machines and contexts can be created and edited using different editors, from the more graphical built-in Event-B editor, containing popup

menus and "wizards", to purely text-based ones such as Camille [31]. The RODIN Platform can be extended with the use of plug-ins that add new features for modelling and proving. In the following, we describe two such plug-ins that we have used in our work.

4.2.1 ProB

ProB [111] is an animator and model checker for a number of formal languages, including the B-Method and Event-B. Animation of an Event-B model refers to "executing" it, in the sense that it is possible to stepwise go through enabled events in a model and examine the changes in the values of variables, i.e., different states. To illustrate why this is useful in addition to proving correctness, we offer the following example.

Consider a random number generator whose specification is to output a number between 0 and 10. If we create a model where the output can be 11, we cannot prove that model correct according to the specification. However, if we create a model where the output is always 4, it does not invalidate the specification and is therefore provably correct, but likely not what we wanted in the first place. Animation helps us find these situations where the model does not do what we intended it to do, by allowing us to explore the possible states of the model.

Model checking is "an automatic technique for verifying finite state concurrent systems" [38]. This is done by automatically exploring (ideally) all the possible (reachable) states of the model, which means that the state space explosion problem is the "main challenge" [38] for model checking. This also explains the need for the restriction to finite state systems. In our work we have not used the model checking aspect of ProB, but the animation part has found use. In Section 5.2.2 we describe how we have used ProB in our work.

4.2.2 Event-B Modularisation Plug-in

In Event-B, there is no built-in approach for creating large models consisting of smaller parts. Therefore, constructing and proving models of such systems can be cumbersome and there have been different approaches to solve this problem [28, 65]. We found that the Modularisation Plug-in [99] offered an approach suitable for our work, in addition to being immediately usable in the form of a plug-in. The Modularisation Plug-in allows the model to include smaller parts, *modules*, of which *instances* can be imported. The modules can then be refined separately and independently from the main model. In our work, this meant that when modelling an overall network structure we could model a network node as a module, and the model of the network would then contain many instances of this module. We will describe this approach further in Section 5.1.

5 | Modelling Media Distribution Algorithms

Having given the background for and methods used in our work, we now turn our attention towards the content of the published articles that form the basis for this dissertation. We envision a complete media distribution system based on a distributed architecture, such as peer-to-peer networking, in which different parts of the system have been formally modelled and verified. Figure 5.1 shows our view of one client in such a system, with our work in different areas indicated by different letters.

As previously mentioned, we focus on three major parts of a media distribution system: network structure and coordination, transport of content over the network, and decoding of encoded content. Our work on network structure and coordination corresponds to the area marked A in Figure 5.1 and is described in Section 5.1. For our work on transport of content over the network we refer to Section 5.2, where we describe our work on content transfer algorithms. This includes both BitTorrent-based on-demand streaming, corresponding to area B in Figure 5.1, and generalised content transfer, corresponding to both areas B and C in Figure 5.1. Finally, in Section 5.3 we describe our work on media decoding, corresponding to area D in Figure 5.1.

5.1 Network Structure and Coordination

As mentioned earlier, a digital media distribution system will by definition involve a computer network of some sort. This means that we will have a set of network nodes that are able communicate with other nodes in order to distribute media content. In our work, we assume that from an application point of view the network topology is not static and pre-defined, and that an application may itself choose which other network nodes it will try to communicate with. As formally modelling such a system in Event-B is not the easiest of tasks, we initially turned our attention towards modelling the algorithms used by a node in a specific type of network, as described in Section 5.2.1.

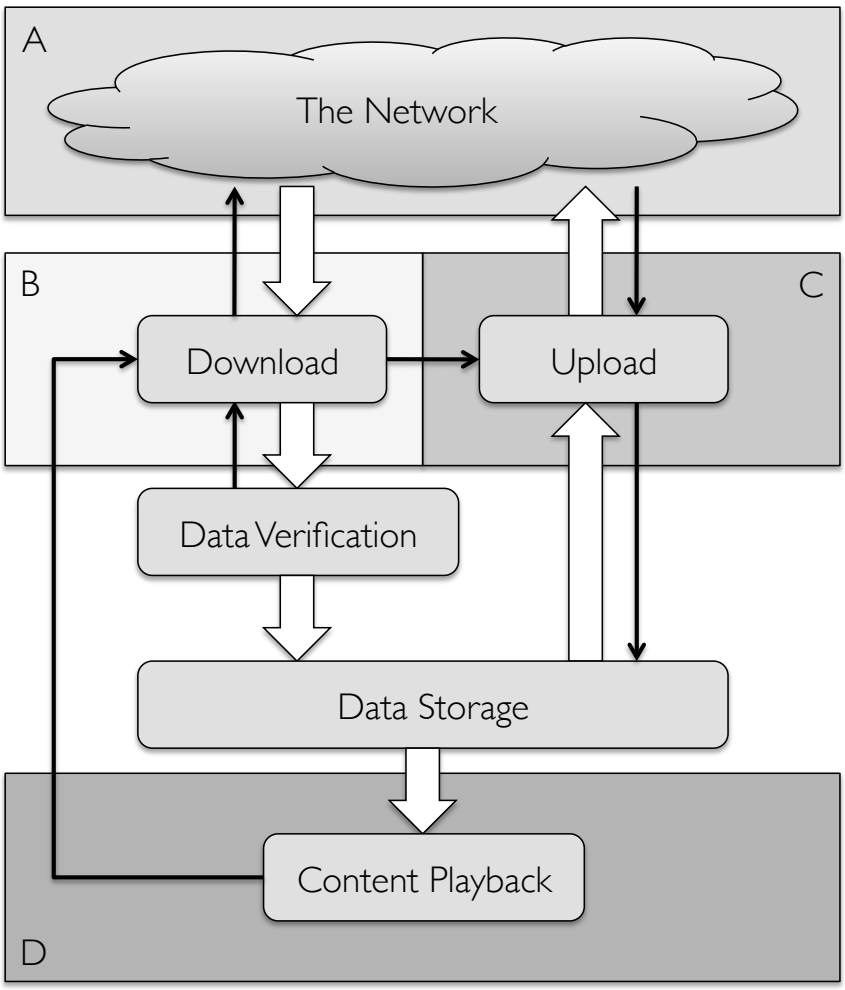


Figure 5.1: Our client-side view of a peer-to-peer-based media distribution system. Wide white arrows indicate flow of data, while thin black arrows indicate flow of control or metadata.

Returning to the task of modelling the network itself, we started abstractly and focused on modelling the inter-peer relations in a peer-to-peer network [107]. This involved thinking about the basics of such a network: peers must be able to connect to other peers in the network, and for that to happen peers need to be aware of other peers. Modelling this in Event-B gave us events for connecting two peers to each other, as well as one peer becoming aware of a subset of other peers, respectively. We then refined this model into a model where the number of connections each peer could have was limited, eliminating the otherwise eventual result of every peer being connected to all other peers. Thus, a connection between two peers

should not always be possible, and this necessitated not only an event for setting the connection limit per peer, but also events for attempting a connection as well as aborting an unsuccessful connection attempt [108]. This model was then further refined by adding information about whether peers are online or not, necessitating events for joining and leaving the (active) network, with the necessary prerequisites that peers must be online before attempting to connect to other peers as well as close all connections before going offline. Additionally, we modelled the situation where a peer may not be able to accept incoming connections. Such a situation is common in many real-life situations [56], for instance when using traditional network address translation (NAT) [43, 132].

Further refinement of a model like this would increase the complexity, and at this stage we therefore involved the Modularisation Plug-in for Event-B. Our idea behind modularising our model was to separate each peer's internal functionality from the functionality of the network as a whole. This would make the peers independent of each other, allowing us to connect this work to our other work where we modelled one specific peer (Paper II, described in Section 5.2.1). This modularisation approach is described in Paper III and further explained in our technical report [108], but here we give a short summary to better connect it with our other work.

Modularisation in Event-B uses an approach in which a part of the model can be separated into a *module*, and events are separated depending on which parts of the model they affect. Any event that affects only the module internally is deemed a *process* and exists only in the module, while events that affect the whole model remain in the overall machine but calls on corresponding *operations* in the module. The module *interface* presents the module to the Event-B machine that imports, or *uses*, the module, while the module *body* is an Event-B machine that implements the operations specified by the interface.

In our work, using this approach means that the only events remaining in the main model are events for connecting and disconnecting peers, and all the other events from previous refinements become processes in a peer module. This may seem counterintuitive, but even leaving the network only directly affects one peer at a time, because in our model it is only possible for a peer to leave the network when the peer has no connections to other peers. Likewise, a peer joining a network does not directly affect any other peer, although it opens the possibility of establishing connections to other peers.

By modelling node coordination in a peer-to-peer network like this, starting from an abstract model and continuing towards a model where the peers are modelled as individual instances of a module, we create a reusable model where the individual parts can be developed separately. Furthermore, the mixed data- and control-oriented coordination approach results in a rather sophisticated coordination in our distributed model.

Work division. The modelling was done by Petter Sandvik, based on an idea by Kaisa Sere. The modularisation approach was suggested by Luigia Petre. Paper III was written mainly by Petter Sandvik, with contributions by Luigia Petre.

5.2 Content Transfer Algorithms

Now that we have a model of a network structure that could be used for our media distribution system, we turn our attention to the act of getting the content across the network. We start by looking at expanding our previous work on adapting peer-to-peer file sharing technology for on-demand streaming [118], beginning with an idea and a simulation in Section 5.2.1 (Paper I) and continuing in Section 5.2.2 with formal modelling and verification of this concept (Paper II). We then continue in Section 5.2.3 with a more general formalisation of content transfer algorithms (Paper V).

5.2.1 BitTorrent Streaming

In 2008, we described an idea of how a BitTorrent-based media streaming system could work [118]. Based on this, we developed the distance-availability weighted piece selection method (DAW) [120]. As described in Section 3.3.1, in BitTorrent the content to be transferred is partitioned into pieces that are requested primarily rarest-first, which works well for situations in which the content is to be transferred as a whole before it is being used. Our suggestion for a different way of requesting pieces comes from the idea that in a streaming situation, it is important to request pieces of content that are close to being played back, but for a BitTorrent-like system to function efficiently with respect to the tit-for-tat mechanism the availability of each piece of content must also be taken into account. Thus, the distance-availability weighted method was born, prioritising each piece based on its distance (i.e., difference in piece number to the piece currently being played back) multiplied by its availability (i.e., number of peers who have that piece). In our work the distance and availability are given equal weights, but giving either one more weight than the other would certainly also be possible, although we have not explored what the practical effect of this would be.

We wrote simple scripts that simulated the piece selection methods but not an actual network, and using these simulations we compared the performance of piece selection methods. In addition to the distance-availability weighted method we also simulated requesting pieces sequentially, as well as requesting using the rarest-first method with buffer (RFB), as described by Vlavianos et al. [145]. These simulations included situations where all peers joined the network simultaneously, as well as situations where the joining was spread

out over time. We found that compared to the two other methods, DAW is better overall for the BitTorrent network than the sequential request method, and at the same time DAW provides better individual performance than RFB [120].

After that, we extended our work with another set of simulations, into which we also included BiToS [145], a BitTorrent piece selection method specifically for streaming. This piece selection method partitions the pieces of content into a small "high-priority" consisting of pieces close to being played back and a much larger "low-priority" set of other pieces, with pieces in the high priority set having a much higher probability of being selected. The probabilistic approach means that unlike the other methods, BiToS cannot guarantee that all pieces will be requested in time for playback even if the download rate is large enough. We therefore performed a set of simulations in which we measured how often media playback would skip and/or stop when using the four mentioned piece selection methods under different conditions. We found that DAW compared favourably to BiToS, although the differences were smaller than we previously believed. This work was published as Paper I.

Work division. The simulations were done by Petter Sandvik. Paper I was written by Petter Sandvik, based on discussions with Mats Neovius.

5.2.2 Formally Modelled BitTorrent Streaming

Continuing on our previous work, we turned our attention towards a formally verified model of the piece selection algorithms. As mentioned previously, we realised that modelling the entire network would be difficult, and therefore focused on modelling how one peer in the peer-to-peer network sees the network. In a sense, we assumed that the peer knows and can adapt its own behaviour, but the behaviour of other peers can be seen as non-deterministic [123].

The full details on how we modelled BitTorrent-based media streaming can be found in our technical report [122] associated with Paper II, but here we shortly describe the process. Our initial, abstract model contained events for selecting pieces of content and playing back a piece of content. In our first refinement, we added information about exactly which pieces were selected, and in the second and third refinement we added information about how many and which pieces have been requested and transferred. In the fourth refinement, we introduced the concept of priority, thereby having a way of specifying which pieces should be selected. In Paper II, we refer to this refinement step as our common model. After that, we refined our model in three different ways, representing different piece selection methods that calculate the priority in different ways. These three methods are the same as in [120]: sequential piece selection, rarest-first method with buffer and the distance-availability weighted method. The previously mentioned and

simulated BiToS [145] uses probabilities, and in Event-B it could therefore not be modelled to the same level as the others, preventing us from including it.

To be able to compare the behaviour of our formal models with our simulations, as described in Paper I, we used ProB [111] to animate our Event-B models. In practice, this meant that we exported our refined model from the RODIN Platform tool into ProB and animated it by generating random traces until we reached a certain state, and then repeated the process a number of times to get the average values of the variables we were interested in. In our case, we were interested in looking at the availability of different content pieces after a certain number of pieces have been selected and transferred. Because of how our formal models were created and the way ProB interacts with them, memory and processing time constraints forced us to use smaller numerical values for the number of peers and pieces of content than we would in a real-life situation. However, we did re-run the simulations from Paper I with these smaller numbers. We were then able to compare the results obtained from simulating a whole network with the average of 40 random animation traces from ProB. In this situation, we found that a formal model of one peer behaved on average very similarly to an average peer in our simulation of a peer-to-peer network. These results were published as Paper II.

Work division. The initial idea was discussed by Petter Sandvik and Kaisa Sere. Formal modelling was done by Petter Sandvik, with help from Marina Waldén. Paper II was written by Petter Sandvik.

5.2.3 Generalised Content Transfer

While working on formally modelling different BitTorrent piece selection algorithms, we noted a lack of standardisation in how these algorithms were described. This resulted in an idea for a specification language for these types of algorithms; a generalisation of this idea to include other types of content transfer algorithms was presented at NWPT 2012 [119].

The idea behind the Specification for Content Transfer Algorithms language (SPECTA) is to be able to specify, in a manner that is easily understandable for both humans and machines, how an algorithm that piece-wise transfers content chooses the order in which pieces of content are transferred. We should then be able to automatically translate from this language into other languages and formalisms, for purposes including validation and code generation.

In SPECTA, we specify the process of choosing which piece to transfer by stacking *selections*. Each selection consists of a *condition* that the system must be in for that selection to be used, as well as *criteria* that an eligible

piece of content must satisfy to be selected for transfer. If the condition is not fulfilled or the condition is fulfilled but there is no piece matching the criteria, the next selection is tried.

To show that a content transfer algorithm written in SPECTA can then be translated and for easy comparison with our previous work, we chose to start by focusing on translating to Event-B. Our simple translator did not translate SPECTA code to Event-B models, but to Event-B code for events that could be inserted into a model. We started by describing and translating DAW [120, 121], and compared the model with events generated from that SPECTA code with our previous models, as described in Section 5.2.2. We found that when the events were generated from SPECTA code the complexity was lower, as evidenced by fewer proof obligations needed and fewer refinement steps used to get the same functionality. This work is presented as Paper V.

Work division. Paper V was written by, as well as based on an original idea by, Petter Sandvik.

5.3 Formally Modelled Media Decoding

Looking at Figure 5.1 and leaving the process of verifying the integrity of the received content, as well as storing it, out of scope, the next step after having received media content is to play it back. This is a very wide topic, but we have focused on one particular area: decoding of video that is encoded using the H.264 [66] standard. Video content consists of a series of images, or *frames*, and the encoding usually takes advantage of the previously mentioned particularity that there is often very little difference between one frame and the next, and thus it can be efficient to store differences between frames rather than the whole image each time. In many video compression formats, including H.264, the frame is divided into smaller square *macroblocks*, and these blocks are encoded individually. H.264 allows macroblocks to be based on other macroblocks with similar content, either in the same frame, in a previous frame, or in a frame that is to be displayed later. This is known as *prediction*. This partition into different types of macroblocks means that we also have different types of frames: I-frames, in which macroblocks refer only to macroblocks in the same frame, P-frames, where macroblocks may also refer to macroblocks in a previous frame, and B-frames, in which macroblocks may refer to macroblocks in the same frame and two other frames. The H.264 standard has several different levels and extensions, and this is therefore only a subset of what is possible to encode with H.264 [135]. Having these different types of frames creates dependencies between them; for instance, a B-frame cannot be decoded until the other frames its blocks depend on have been decoded. This creates a problem when we are trying to decode efficiently, for instance when trying to decode frames in parallel.

We started modelling these dependency relations in Event-B by having an abstract model of a set of unprocessed frames, and an event that processes them. In the following two refinement steps, we added the different types of frames and their dependencies (or lack thereof, in the case of I-frames), as well as introduced the concept of macroblocks with the notion that frames are not processed all at once but block by block. In the third refinement, we added the macroblocks and their dependency relations, and in the fourth refinement step we refined the events to build up a stream of output frames [94].

Extending our previous work, we turned towards the idea of specifying how the decoding process could be done in parallel. The main ways of doing this is *functional partitioning*, where different parts of the decoding process are run in parallel, and *data partitioning*, in which the same part of the decoding process is run in parallel with different data. The latter has been found to have better scalability [141], and we therefore decided to focus on data partitioning. The two main ways of doing this for H.264 encoded video are *frame-level parallelisation* and *macroblock-level parallelisation*, specifying whether the data units to be processed in parallel are the whole frames or individual macroblocks, respectively. When doing frame-level parallelisation the processing of macroblocks within a frame is done in-order, but for macroblock-level parallelisation this is not the case. Thus, for macroblock-level parallelisation we need to take into account the constraint that a block within a frame can depend on blocks to its left, top left, top, and top right positions.

We refined our previous formal model in two different ways, representing the two different parallelisation techniques. Common for both was having a set of *processing units*, of which each could be assigned to at most one data unit at a time. We noted that the macroblock-level parallelisation required more proofs than frame-level parallelisation, mainly because of the complexity of handling individual blocks rather than frames, as well as the handling of several special cases. The latter primarily refers to when macroblocks are located at the edge of a frame and thus, compared to the common case, fewer surrounding blocks can be used for prediction. This work was published as Paper IV.

Work division. Initial modelling was done by Kristian Lumme, based on an idea by Luigia Petre and Kaisa Sere, with further modelling done by Petter Sandvik. Paper IV was written mainly by Petter Sandvik, with contributions from Kristian Lumme and Luigia Petre.

6 | Related Work

As with most research, our work does not exist in a vacuum of its own, but relies on previous work as its foundation – also referred to as "standing on the shoulders of giants"[1]. In this chapter we look at previous work related to the problems we have aimed to solve in our research. We start by looking at networks and how they have been modelled, in particular formally modelled. We continue by looking at previous work on media distribution, including streaming using peer-to-peer networks and related technologies. We also describe related approaches to media encoding and decoding, with a focus on parallel processing.

6.1 Modelling Networks

With networks being of such importance as they are in today's world, it should come as no surprise that there has been a wide variety of research into networks. Our work has focused on the formal modelling of overlay (application-level) networks, but in the following we will give a few examples of other research into networks, with an emphasis on formal modelling.

There are many different ways of classifying networks, and one way is to separate *dynamic* and *static* networks. The former refers to networks with a varying set of nodes and interconnections, while the latter refers to networks with a fixed set of nodes and interconnections [70]. Our focus has been on dynamic networks, as peer-to-peer networks and other loosely formed overlay networks are of this type. An overview of work in formal modelling of dynamic and static networks has been given by Kamali [70].

Networks of different scale appear in different areas of computing. For instance, networks-on-chip (NoC) are a way of using networking methods to coordinate components of a single microchip, and Ostroumov et al. have formally modelled the dynamic reconfiguration of such a network [105]. By stacking multiple layers of 2D NoCs it is possible to make a 3D NoC, and the high complexity and reliability requirements of such a design have encouraged

[1] This quote has often been attributed to Isaac Newton, and although Newton did use the expression, he is not the originator (as explored by Merton [97]).

the use formal methods to model 3D NoCs [73, 74]. An example of a larger network type that has been of scientific interest is wireless networking [7, 8, 34]. In particular, wireless sensor networks [78, 116] and wireless sensor-actor networks [71, 72] have been formally modelled and analysed.

Moving towards the application domain and distributed protocols of networks, Bochmann and Sunshine noted as early as 1980 that in the design of communication protocols it would be useful to involve formal methods, including the concepts of specification, abstraction and refinement [24]. An early unified approach for formally describing and validating several distributed network protocols was presented by Segall [127]. Closer to present day, Bhargavan et al. have used a theorem prover and a model checker to prove key properties of distance vector routing protocols [21], including the Ad-Hoc On-Demand Distance Vector (AODV) protocol. Höfner and McIver have used statistical model checking to verify the AODV protocol for networks of up to 100 nodes [60], while Fehnker et al. extended this approach with a mobility model [51].

When it comes to peer-to-peer networks, BitTorrent has been modelled for performance analysis by Qiu and Srikant [112]. The distributed hash table Chord has been formally modelled by Zave [153]. An architectural model for peer-to-peer networks has been formalised by Yan [151], based on the Action Systems [10, 12, 146] formalism and Gnutella [114] peer-to-peer network. The combination of formal methods and networks has also found practical use, for instance in Amazon Web Services, as described by Newcombe et al. [104].

6.2 Network Media Distribution

In Section 3.5.3, we mentioned that Spotify has used both its own network of servers and a peer-to-peer network to deliver content. This hybrid approach has been suggested before, for instance by Xu et al. [149]. The idea behind such an approach is to strike a balance between the dedicated nodes in a CDN, which are reliable and have low latencies but are comparatively expensive to deploy and maintain, and the nondedicated peers in a peer-to-peer network, which cost nothing to deploy but are less reliable and have higher latencies.

As mentioned in Section 3.3.1, there have been many proposals as to how the BitTorrent piece selection methods could be adapted to on-demand streaming [32, 33, 103, 120, 129, 145]. In practice, requesting the pieces sequentially seems to be the prevalent method [23, 57], although the method proposed by Mol et al. [103] is also used [154]. A comparison of sequential and rarest-first piece selection methods for BitTorrent streaming has been done by Parvez et al. [106]. A more comprehensive comparison of different BitTorrent-like approaches for on-demand video streaming has been done by D'Acunto et al. [42], focusing on three different methods for piece selection.

The first approach, *window-based piece selection*, is based on that of Shah and Pâris [129], and uses a sliding window of a certain number of pieces from which pieces are selected, usually rarest-first. This approach has the disadvantage that the tit-for-tat incentive mechanism only works with peers whose sliding windows are overlapping. The second approach, called *probabilistic piece selection*, is based on that of Carlsson et al. [33], and uses Zipf probability such that the lowest numbered piece not downloaded by the peer is given the highest probability and other pieces are given progressively lower probability of being selected. The idea here is similar to our distance-availability weighted method in Paper I, in that not only should pieces close to playback be selected – which benefits the peer itself – but also pieces that are further away, for the benefit of the network overall. The third approach, called *priority-based piece selection* by D'Acunto et al., is based on an idea by Mol et al. [103] and similar to the rarest-first method with buffer described by Vlavianos et al. [145]: A specific number of pieces close to being played back are requested in-order, and when all those pieces have been selected, rarest-first is used to select among remaining pieces. The method by Mol et al. further splits the remaining pieces into a "mid priority" set and a "low priority" set, with all pieces in the former being selected before any pieces in the latter are considered.

The mentioned work has focused on on-demand streaming, but the important use of live streaming services such as Bambuser for reporting of occurring news and events [46] has shown the importance of as well as need for live streaming. Mol et al. have studied BitTorrent for live streaming [102], as have Tewari and Kleinrock [139], with the version by Mol et al. being implemented in practice [154].

Finally, as data sizes and speeds continue to grow, Tierney et al. have showed that there is a need for more efficient transfer protocols for big data [140], and Tolia et al. have described a BitTorrent-influenced approach to data transfer in general [142].

6.3 Encoding and Decoding of Media

The optimisation of video encoding has been a popular research subject. As early as 1998, Yung and Chu described how H.261 video encoding can be done in parallel by workload balancing [152]. More recently, Sun et al. showed how this can be done for efficient parallel H.264 encoding [136]. Li et al. have showed how a video encoder can be parallelised on a bus network by pre-partitioning the data such that no inter-processor communication is needed [93]. We mentioned 3D networks-on-chip in Section 6.1, and Xu et al. have showed how parallel H-264 encoding can be optimised for a 3D NoC [150].

The H.264 decoding process is a part of the encoding process [150], but dedicated video decoding has also been an important research topic,

particularly with regards to parallelisation. One can distinguish between *functional partitioning*, where different parts of the decoding process are run in parallel, and *data partitioning*, where the same process is done in parallel but on different data. Van der Tol have compared these two, and found greater scalability in the latter [141]. Chi and Juurlink have combined both functional partitioning and data partitioning in order to achieve even greater scalability [36]. For H.264 decoding, the two main ways of data partitioning are frame-level parallelisation and macroblock-level parallelisation, both of which we discussed in Section 5.3. Mesa et al. have formally described the limit for the improvement in performance by macroblock-level parallelisation [98], and Meenderinck et al. have combined frame- and macroblock-level parallelisation to a "Dynamic 3D-Wave" parallelisation strategy that could potentially process hundreds of macroblocks in parallel even on low-end devices [96].

Decoding video in parallel can improve not only performance but also energy efficiency, as shown in the work by Kılıçarslan et al. [81] and Seitner et al. [128]. Further decrease in energy usage can be achieved by not running the processor cores at a higher speed than what is necessary for decoding the media, and this run-time management of media decoding has been formally modelled by Salehi Fathabadi et al. [117].

7 | Conclusion

In this chapter we overview our research contributions, as described earlier in this dissertation, and present the conclusions we have reached based on the work we have undertaken so far. We also outline possible further work directions and challenges within the subject of digital media distribution.

7.1 Research Contributions

The process of media distribution has come a long way from its origins. To transfer media from one place to another we no longer need to move physical objects around, and if we still choose to do so, those objects more often than not contain abstract mathematical representations of phenomena that we cannot experience ourselves without some form of computer software and hardware translating for us. In other words, software running on some form of computer is required whether we get the latest media content through a network or on a physical medium such as a Blu-ray disc. The software algorithms involved in the process of getting digital media from some other location and into a format presentable on a user's device has been the main subject of our work.

Digital media distribution systems already exist, and their continued, and in many cases increasing, usage shows that they work in practice. In this thesis, our aim has been to show that these systems can be improved upon, and that improvements in performance, efficiency and and availability can be done in conjunction with improvements in software quality and reliability through the use of formal methods.

Starting with the network itself, we have showed how a network can be formally constructed and modularised in Event-B, such that we can separate the modelling of one node from the modelling of the network itself. This model therefore includes both the network coordination and the functionality of each network node as separate entities, allowing us to focus on the latter in our following work.

Focusing on the functionality of each node in a network, we have showed how the piece selection algorithm used by BitTorrent can be adapted for

on-demand media streaming, and how this can be modelled in Event-B. We have also showed that modelling one peer in Event-B and animating it in ProB gives results similar to simulating an entire network of peers, while allowing us to prove certain properties mathematically.

Going beyond our idea of on-demand streaming based on peer-to-peer file sharing, we have showed that having a formal specification language for content transfer algorithms can make these easier to understand. We have also showed that generating Event-B code from this language can result in less complexity compared to models based on written specifications, which improves the provability of the formal model and helps to ease the development.

In the final software step of bringing media to people, we focused on video decoding, and in particular how it could be done more efficiently. We have showed that video decoding can be done in parallel based on formally defined dependencies between frames and blocks in a video sequence. This means that also this step can be performed in a way that is proven correct.

Overall, we have presented a view of how different parts of a digital media distribution can be formally modelled and verified. These algorithms and methods that we have chosen all have a basis in real-world examples, thus bridging the all too frequently existing gap between industrial practices and academic research.

7.2 Future Work and Challenges

Given the increasing importance of digital media distribution, it is evident that there is still much to be done research-wise in this field. In the following, we present a few challenges and possible directions for future work.

One area that is especially important to look at is the increasing use of mobile devices. These devices often depend on cellular networks, and especially in developing countries the only commonly available Internet access may be through the cellular network. In a traditional landline-based Internet connection, no matter which technology is used, the "last mile" – i.e., the traffic between the Internet service provider (ISP) and the end user – is "free", in the sense that it does not incur any additional cost, while the traffic from the ISP towards the rest of the Internet is what the ISP pays for. Also, when transferring content to the end user, the end user's uplink is mostly unused, and can therefore be used for instance for uploading data to other peers in a peer-to-peer network. In contrast, the traffic from the ISP to the end user in a cellular network is not "free", in the sense that there are many devices that compete for the bandwidth. Also, uploading data will in many cases reduce the available bandwidth for downloading. In summary, this means that for cellular networks, the virtual bottleneck becomes the connection between

the ISP and the end user, rather than between the ISP and the rest of the Internet.

There are several implications from this move towards cellular and mobile data. One is that it is in the interest of the end user to lower the data rates needed for the media content; not so much because of transfer speeds, which are increasing, but mainly because of limitations on the total amount of data transfer allowed. Additionally, there needs to be a balance between how much content is transferred in advance; buffering lots of data that may never be needed wastes bandwidth, and buffering too little can degrade media playback in changing conditions. When it comes to lowering the data rate of the content, initiatives such as the High Efficiency Video Coding (HEVC) [134] have been launched to decrease the amount of data required, with HEVC aiming to decrease the amount of data required by 50% compared to its predecessor H.264.

Considering our own work, one obvious direction to continue would be to implement our findings in actual, end-user software. However, when considering on-demand media streaming, it seems as though it is being limited not only by the technical challenges, but also by the lack of easy, legal availability of content. The recent success of the Popcorn Time application [57], whose developers conveniently seem to ignore the legal aspects of its use, has showed that there is public interest in a decentralised, collaborative media distribution system. Our hope is that a system based on our work, with proven correctness and reliability, could find approval also from content producers and rights holders.

We believe that the work described in this thesis can be useful also outside of digital media distribution. Part of what we have done is to make content transfer formally correct and extend it to include a timewise ordering of data. Therefore, our work could find use in areas such as gathering of sensor data, especially in such cases where the sensors already form a peer-to-peer like network. Moreover, energy consumption is an increasingly important topic that we would like to focus on in our upcoming work, beyond the media decoding aspects.

Bibliography

[1] Kota Abe, Tatsuya Ueda, Masanori Shikano, Hayato Ishibashi, and Toshio Matsuura. Toward Fault-tolerant P2P Systems: Constructing a Stable Virtual Peer from Multiple Unstable Peers. In Antonio Liotta, Nick Antonopoulos, George Exarchakos, and Takahiro Hara, editors, *Proceedings of The First International Conference on Advances in P2P Systems (AP2PS 2009)*, pages 104–110. IEEE Computer Society, 2009.

[2] Jean-Raymond Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[3] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[4] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farad Mehta, and Laurent Voisin. Rodin: An Open Toolset for Modelling and Reasoning in Event-B. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(6):447–466, 2010.

[5] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, and Laurent Voisin. An Open Extensible Tool Environment for Event-B. In Zhiming Liu and Jifeng He, editors, *Formal Methods and Software Engineering. 8th International Conference on Formal Engineering Methods, ICFEM 2006. Proceedings*, volume 4260 of *Lecture Notes in Computer Science*, pages 588–605, November 2006.

[6] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Matteo Varvello, Volker Hilt, Moritz Steiner, and Zhi-Li Zhang. Unreeling Netflix: Understanding and Improving Multi-CDN Movie Delivery. In *2012 IEEE INFOCOM Proceedings*, pages 1620–1628. IEEE, March 2012.

[7] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38:393–422, 2002.

[8] Ian F. Akyildiz, Xudong Wang, and Weilin Wang. Wireless mesh networks: a survey. *Computer Networks*, 47:445–487, 2005.

[9] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Computing Surveys*, 36(4):335–371, December 2004.

[10] Ralph-Johan Reinhold Back and Reino Kurki-Suonio. Decentralization of process nets with centralized control. In *Proceedings of the 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 131–142, 1983.

[11] Ralph-Johan Reinhold Back and Kaisa Sere. Stepwise Refinement of Action Systems. *Structured Programming*, 12(1):17–30, 1991.

[12] Ralph-Johan Reinhold Back and Kaisa Sere. From Action Systems to Modular Systems. *Software - Concepts and Tools*, 17(1):26–39, 1996.

[13] Ralph-Johan Reinhold Back and Kaisa Sere. Superposition Refinement of Reactive Systems. *Formal Asp. Comput.*, 8(3):324–346, 1996.

[14] Peter Backx, Tim Wauters, Bart Dhoedt, and Piet Demeester. A comparison of peer-to-peer architectures. In *Eurescom Summit*, 2002.

[15] Frédéric Badeau and Arnaud Amelot. Using B as a High Level Programming Language in an Industrial Project: Roissy VAL. In Helen Treharne, Steve King, Martin Henson, and Steve Schneider, editors, *ZB 2005: Formal Specification and Development in Z and B*, volume 3455 of *Lecture Notes in Computer Science*, pages 334–354. Springer Berlin Heidelberg, 2005.

[16] Jim Bankoski, Paul Wilkins, and Yaowu Xu. Technical Overview of VP8, an Open Source Video Codec for the Web. In *2011 IEEE International Conference on Multimedia and Expo (ICME 2011)*, pages 1–6. IEEE Computer Society, July 2011.

[17] Rodrigo Barbosa, Carlos Kamienski, Dênio Mariz, Arthur Callado, Stênio Fernandes, and Djamel Sadok. Performance evaluation of P2P VoIP applications. In *17th International workshop on Network and Operating Systems Support for Digital Audio & Video (NOSSDAV'07)*, June 2007.

[18] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for Agile Software Development. `http://agilemanifesto.org`, 2001. (Accessed October 2015).

[19] Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier. Météor: A Successful Application of B in a Large Project. In Jeannette M. Wing, Jim Woodcock, and Jim Davies, editors, *FM'99 – Formal Methods*, volume 1708 of *Lecture Notes in Computer Science*, pages 369–387. Springer Berlin Heidelberg, 1999.

[20] Thomas E. Bell and T. A. Thayer. Software Requirements: Are They Really a Problem? In *Proceedings of the 2nd International Conference on Software Engineering (ICSE '76)*, pages 61–68. IEEE Computer Society Press, 1976.

[21] Karthikeyan Bhargavan, Davor Obradociv, and Carl A. Gunter. Formal Verification of Standards for Distance Vector Routing Protocols. *Journal of the ACM*, 49(4):538–576, July 2002.

[22] Kenneth P. Birman. *Guide to Reliable Distributed Systems: Building High-Assurance Applications and Cloud-Hosted Services.* Springer-Verlag London Limited, 2012.

[23] BitTorrent | How does Streaming work? `http://help.utorrent.com/customer/portal/articles/1766517-how-does-streaming-work-`, November 2014. (Accessed October 2015).

[24] Gregor V. Bochmann and Carl A. Sunshine. Formal Methods in Communication Protocol Design. *IEEE Transactions on Communications*, 28(4):624–631, April 1980.

[25] Karlheinz Brandenburg. MP3 and AAC Explained. In *AES 17th International Conference on High-Quality Audio Coding.* AES, August 1999.

[26] Jon Brodkin. Netflix researching "large-scale peer-to-peer technology" for streaming. `http://arstechnica.com/information-technology/2014/04/netflix-researching-large-scale-peer-to-peer-technology-for-streaming/`, April 2014. (Accessed October 2015).

[27] Jon Brodkin. Netflix's many-pronged plan to eliminate video playback problems. `http://arstechnica.com/information-technology/2014/05/netflixs-many-pronged-plan-to-eliminate-video-playback-problems/`, May 2014. (Accessed October 2015).

[28] Michael Butler. Decomposition Structures for Event-B. In Michael Leuschel and Heike Wehrheim, editors, *Integrated Formal Methods. 7th International Conference, IFM 2009, Düsseldorf, Germany, February 16-19, 2009. Proceedings*, volume 5423 of *Lecture Notes in Computer Science*, pages 20–38. Springer Berlin Heidelberg, 2009.

[29] John N. Buxton and Brian Randell, editors. *Software Engineering Techniques – Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969*, April 1970.

[30] Cristiano Calcagno, Dino Distefano, Jeremy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. Moving Fast with Software Verification. `https://research.facebook.com/publications/422671501231772/moving-fast-with-software-verification/`, February 2015. (Accessed October 2015).

[31] Camille Editor. `http://wiki.event-b.org/index.php/Camille_Editor`. (Accessed October 2015).

[32] Niklas Carlsson and Derek L. Eager. Peer-Assisted On-Demand Streaming of Stored Media Using BitTorrent-Like Protocols. In Ian F. Akyildiz, Raghupathy Sivakumar, Eylem Ekici, Jaudelice Cavalcante de Oliveira, and Janise McNair, editors, *NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet. 6th International IFIP-TC6 Networking Conference. Proceedings*, volume 4479 of *Lecture Notes in Computer Science*, pages 570–581. Springer Berlin Heidelberg, May 2007.

[33] Niklas Carlsson, Derek L. Eager, and Anirban Mahanti. Peer-assisted On-demand Video Streaming with Selfish Peers. In *NETWORKING 2009: 8th International IFIP-TC 6 Networking Conference, Aachen, Germany, May 11-15, 2009. Proceedings*, volume 5550 of *Lecture Notes in Computer Science*, pages 586–599, May 2009.

[34] Andrea Cerone, Matthew Hennessy, and Massimo Merro. Modelling MAC-Layer Communications in Wireless Systems. In Rocco De Nicola and Christine Julien, editors, *Coordination Models and Languages. 15th International Conference, COORDINATION 2013, Held as Part of the 8th International Federated Conference on Distributed Computing Techniques, DisCoTec 2013, Florence, Italy, June 3-5, 2013. Proceedings*, volume 7890 of *Lecture Notes in Computer Science*, pages 16–30, June 2013.

[35] Joachim Charzinski. Traffic Properties, Client Side Cachability and CDN Usage of Popular Web Sites. In Bruno Müller-Clostermann, Klaus Echtle, and Erwin P. Rathgeb, editors, *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance. 15th International GI/ITG Conference, MMB&DFT 2010, Essen, Germany, March 15-17, 2010. Proceedings*, volume 5987 of

Lecture Notes in Computer Science, pages 136–150. Springer Berlin Heidelberg, March 2010.

[36] Chi Ching Chi and Ben Juurlink. A QHD-Capable Parallel H.264 Decoder. In *Proceedings of the International Conference on Supercomputing (ICS '11)*, pages 317–326. ACM, May 2011.

[37] Kiho Choi and Euee S. Jang. Royalty-Free Video Coding Standards in MPEG. *IEEE Signal Processing Magazine*, 31(1):145–155, January 2014.

[38] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. The MIT Press, December 1999.

[39] Bram Cohen. BitTorrent - a new P2P app. Yahoo Groups, `https://groups.yahoo.com/neo/groups/decentralization/conversations/messages/3160`. (Accessed October 2015).

[40] Bram Cohen. Incentives Build Robustness in BitTorrent. In *1st Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[41] Tom Cook. A Day in the Life of Facebook Operations. Velocity 2010. `https://www.youtube.com/watch?v=T-Xr_PJdNmQ`. (Accessed October 2015).

[42] Lucia D'Acunto, Nitin Chiluka, Tamás Vinkó, and Henk J. Sips. BitTorrent-like P2P approaches for VoD: A comparative study. *Computer Networks*, 57(5):1253–1276, 2013.

[43] Lucia D'Acunto, Michel Meulpolder, Rameez Rahman, Johan A. Pouwelse, and Henk J. Sips. Modeling and Analyzing the Effects of Firewalls and NATs in P2P Swarming Systems. In *IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum (IPDPSW) 2010*, pages 1–8. IEEE, April 2010.

[44] Deploy – Industrial deployment of system engineering methods providing high dependability and productivity. `http://www.deploy-project.eu/`. (Accessed October 2015).

[45] Emil Dumić, Mario Muštra, Sonja Grgić, and Goran Gvozden. Image Quality of 4:2:2 and 4:2:0 Chroma Subsampling Formats. In Mislav Grgić, Jelena Božek, and Sonja Grgić, editors, *51st International Symposium ELMAR-2009*, pages 19–24. Croatian Society Electronics in Marine – ELMAR, Zadar, September 2009.

[46] Alexandra Dunn. Unplugging a Nation: State Media Strategy During Egypt's January 25 Uprising. *The Fletcher Forum of World Affairs*, 35:15–24, Summer 2011.

[47] Eclipse project. http://projects.eclipse.org/projects/eclipse. (Accessed October 2015).

[48] Ernesto. Spotify Starts Shutting Down Its Massive P2P Network. http://torrentfreak.com/spotify-starts-shutting-down-its-massive-p2p-network-140416/, April 2014. (Accessed October 2015).

[49] Ernesto. How a Private 'Anime' Torrent Tracker Became an Essential Tool for Facebook. http://torrentfreak.com/private-anime-torrent-tracker-became-essential-tool-facebook-150216/, February 2015. (Accessed October 2015).

[50] Event-B and the Rodin Platform. http://www.event-b.org/. (Accessed October 2015).

[51] Ansgar Fehnker, Peter Höfner, Maryam Kamali, and Vinay Mehta. Topology-Based Mobility Models for Wireless Networks. In Kaustubh Joshi, Markus Siegle, Mariëlle Stoelinga, and Pedro R. D'Argenio, editors, *Quantitative Evaluation of Systems. 10th International Conference, QEST 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*, volume 8054 of *Lecture Notes in Computer Science*, pages 389–404. Springer Berlin Heidelberg, August 2013.

[52] Hendrik Fuchs. Improving Joint Stereo Audio Coding by Adaptive Inter-channel Prediction. In *1993 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. Final Program and Paper Summaries.*, pages 39–42. IEEE, October 1993.

[53] Susan Gerhart, Dan Craigen, and Ted Ralston. Case study: Paris Metro Signaling System. *IEEE Software*, 11(1):32–35, 1994.

[54] Global Internet Phenomena Report 1H 2014. https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/1h-2014-global-internet-phenomena-report.pdf, May 2014. (Accessed October 2015).

[55] Global Internet Phenomena Report 2H 2014. https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/2h-2014-global-internet-phenomena-report.pdf, September 2014. (Accessed October 2015).

[56] Mikael Goldmann and Gunnar Kreitz. Measurements on the Spotify Peer-Assisted Music-on-Demand Streaming System. In *IEEE International Conference on Peer-to-Peer Computing (P2P'11)*, pages 206–211. IEEE, August 2011.

[57] Andy Greenberg. Inside Popcorn Time, the Piracy Party Hollywood Can't Stop. http://www.wired.com/2015/03/inside-popcorn-time-piracy-party-hollywood-cant-stop/, March 2015. (Accessed October 2015).

[58] David Hands and Miles Wilkins. A Study of the Impact of Network Loss and Burst Size on Video Streaming Quality and Acceptability. In Michel Diaz, Philippe Owezarski, and Patrick Sénac, editors, *Interactive Distributed Multimedia Systems and Telecommunication Services. 6th International Workshop, IDMS'99 Toulouse, France, October 12–15, 1999 Proceedings*, volume 1718 of *Lecture Notes in Computer Science*, pages 45–57. Springer Berlin Heidelberg, October 1999.

[59] Robert Harley. An LP Primer: How the LP Works. *The Absolute Sound* June/July 2007.

[60] Peter Höfner and Annabelle McIver. Statistical Model Checking of Wireless Mesh Routing Protocols. In Guillaume Brat, Neha Rungta, and Arnaud Venet, editors, *NASA Formal Methods. 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings*, volume 7871 of *Lecture Notes in Computer Science*, pages 322–336. Springer Berlin Heidelberg, May 2013.

[61] How Consumers Judge Their Viewing Experience: The Business Implications of Sub-Par OTT Service. http://www.conviva.com/csr-2015/, 2015. (Accessed October 2015).

[62] Philip N. Howard and Muzammil M. Hussain. The Role of Digital Media. *Journal of Democracy*, 22(3):35–48, July 2011.

[63] Lican Huang. Instant Messaging Based on Semantic P2P Network. In *Fourth International Conference on Networking and Distributed Computing (ICNDC 2013)*, pages 33–35. IEEE, December 2013.

[64] Ming Huo, June Verner, Liming Zhu, and Muhammad Ali Babar. Software Quality and Agile Methods. In *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)*, pages 520–525. IEEE, September 2004.

[65] Alexei Iliasov, Elena Troubitsyna, Linas Laibinis, Alexander Romanovsky, Kimmo Varpaaniemi, Dubravka Ilic, and Timo Latvala. Supporting Reuse in Event B Development: Modularisation Approach. In Marc Frappier, Uwe Glässer, Sarfraz Khurshid, Régine Laleau, and Steve Reeves, editors, *Abstract State Machines, Alloy, B and Z. Second International Conference, ABZ 2010, Orford, QC, Canada, February*

22-25, 2010. Proceedings, volume 5977 of *Lecture Notes in Computer Science*, pages 174–188. Springer Berlin Heidelberg, 2010.

[66] ITU-T. H.264: Advanced video coding for generic audiovisual services. Recommendation 02/2014. `http://www.itu.int/rec/T-REC-H.264-201402-I/en`. (Accessed October 2015).

[67] Mikel Izal, Guillaume Urvoy-Keller, Ernst W. Biersack, Pascal Felber, Anwar Al Hamra, and Luis Garcés-Erice. Dissecting BitTorrent: Five Months in a Torrent's Lifetime. In Chadi Barakat and Ian Pratt, editors, *Passive and Active Network Measurement. 5th International Workshop, PAM 2004. Proceedings*, volume 3015 of *Lecture Notes in Computer Science*, pages 1–11. Springer Berlin Heidelberg, 2004.

[68] James D. Johnston. Perceptual Transform Coding of Wideband Stereo Signals. In *1989 International Conference on Acoustics, Speech, and Signal Processing (ICASSP-89)*, pages 1993–1996. IEEE, May 1989.

[69] Hari Kalva and Jae-Beom Lee. The VC-1 Video Coding Standard. *IEEE MultiMedia*, 14(4):88–91, October 2007.

[70] Maryam Kamali. *Reusable Formal Architectures for Networked Systems*. PhD thesis, Åbo Akademi University, September 2013. TUCS Dissertations Number 162.

[71] Maryam Kamali, Linas Laibinis, Luigia Petre, and Kaisa Sere. Self-Recovering Sensor-Actor Networks. In Mohammad Reza Mousavi and Gwen Salaün, editors, *Ninth International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA 2010)*, volume 30 of *Electronic Proceedings in Theoretical Computer Science*, pages 47–61, September 2010.

[72] Maryam Kamali, Linas Laibinis, Luigia Petre, and Kaisa Sere. Formal development of wireless sensor–actor networks. *Science of Computer Programming*, 80(A):25–49, February 2014.

[73] Maryam Kamali, Luigia Petre, Kaisa Sere, and Masoud Daneshtalab. Formal Modeling of Multicast Communication in 3D NoCs. In *2011 14th Euromicro Conference on Digital System Design (DSD)*, pages 634–642. IEEE, September 2011.

[74] Maryam Kamali, Luigia Petre, Kaisa Sere, and Masoud Daneshtalab. Refinement-Based Modeling of 3D NoCs. In Farhad Arbab and Marjan Sirjani, editors, *Fundamentals of Software Engineering. 4th IPM International Conference, FSEN 2011, Tehran, Iran, April 20-22, 2011, Revised Selected Papers*, volume 7141 of *Lecture Notes in Computer Science*, pages 236–252. Springer Berlin Heidelberg, April 2012.

[75] Jussi Kangasharju, Keith W. Ross, and Jim W. Roberts. Performance evaluation of redirection schemes in content distribution networks. *Computer Communications*, 24(2):207–214, 2001.

[76] Schmuel Katz. A Superimposition Control Construct for Distributed Systems. *ACM Transactions on Programming Languages and Systems*, 15:337–356, April 1993.

[77] Sebastian Kaune, Jan Stolzenburg, Aleksandra Kovačević, Ralf Steinmetz, and Ruben Cuevas. Understanding BitTorrent's Suitability in Various Applications and Environments. In *The Third International Multi-Conference on Computing in the Global Information Technology, 2008. ICCGI '08.*, pages 256–262. IEEE, August 2008.

[78] Fatemeh Kazemeyni, Olaf Owe, Einar Broch Johnsen, and Ilangko Balasingham. Learning-based Routing in Mobile Wireless Sensor Networks: Applying Formal Modeling and Analysis. In *IEEE 14th International Conference on Information Reuse and Integration (IRI) 2013*, pages 504–511. IEEE, August 2013.

[79] Gertjan Keesman, Robert Hellinghuizen, Fokke Hoeksema, and Geert Heideman. Transcoding of MPEG bitstreams. *Signal Processing: Image Communication*, 8(6):481–500, 1996.

[80] Ajab Khan, Reiko Heckel, Paolo Torrini, and István Ráth. Model-Based Stochastic Simulation of P2P VoIP Using Graph Transformation System. In Khalid Al-Begain, Dieter Fiems, and William J. Knottenbelt, editors, *Analytical and Stochastic Modeling Techniques and Applications. 17th International Conference, ASMTA 2010, Cardiff, UK, June 14-16, 2010. Proceedings*, volume 6148 of *Lecture Notes in Computer Science*, pages 204–217. Springer Berlin Heidelberg, June 2010.

[81] Damla Kılıçarslan, C. Göktuğ Gürler, Öznur Özkasap, and A. Murat Tekalp. Energy Efficient Video Decoding on Multi-Core Devices. In *Proceedings of the 2nd International Conference on Energy-Efficient Computing and Networking (e-Energy '11)*, pages 63–66. ACM, May 2011.

[82] Eric Klinker. Twitter using BitTorrent on the Backend. `http://blog.bittorrent.com/2010/02/09/twitter-using-bittorrent-on-the-backend/`, February 2010. (Accessed October 2015).

[83] Gunnar Kreitz. Spotify – Behind the Scenes. PowerPoint presentation at KTH, April 28 2011. `http://www.scribd.com/doc/56651812/kreitz-spotify-kth11`. (Accessed October 2015).

[84] Gunnar Kreitz and Fredrik Niemelä. Spotify – Large Scale, Low Latency, P2P Music-on-Demand Streaming. In *IEEE Tenth International Conference on Peer-to-Peer Computing (P2P'10)*, pages 1–10. IEEE, August 2010.

[85] Balachander Krishnamurthy, Craig Willis, and Yin Zhang. On the Use and Performance of Content Distribution Networks. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement (IMW'01)*, pages 169–182, 2001.

[86] Michal Kryczka, Rubén Cuevas, Carmen Guerrero, Arturo Azcorra, and Angel Cuevas. Measuring the BitTorrent Ecosystem: Techniques, Tips, and Tricks. *IEEE Communications Magazine*, 49(9):144–152, September 2011.

[87] Roslyn Layton. Netflix Comes to the Nordics: Lessons in OTT Video. *Nordic and Baltic Journal of Information and Communications Technologies*, 2014(1):109–138, 2015.

[88] Gary T. Leavens, Jean-Raymond Abrial, Don S. Batory, Michael J. Butler, Alessandro Coglio, Kathi Fisler, Eric C.R. Hehner, Cliff B. Jones, Dale Miller, Simon L. Peyton-Jones, Murali Sitaraman, Douglas R. Smith, and Aaron Stump. Roadmap for Enhanced Languages and Methods to Aid Verification. In *Fifth International Conference on Generative Programming and Component Engineering (GPCE'06)*, pages 221–236. ACM, October 2006.

[89] Arnaud Legout, Guillaume Urvoy-Keller, and Pietro Michiardi. Rarest First and Choke Algorithms Are Enough. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (IMC'06)*, pages 203–216, 2006.

[90] Nathaniel Leibowitz, Matei Ripeanu, and Adam Wierzbicki. Deconstructing the Kazaa Network. In *Proceedings of The Third IEEE Workshop on Internet Applications (WIAPP'03)*, pages 112–120. IEEE, June 2003.

[91] Jenq-Shiou Leu and Ming-Hung Huang. Comparison of Piece-based File Sharing Schemes over a Peer-to-Peer Network in a Heterogeneous Network Environment. In *Global Telecommunications Conference 2009 (GLOBECOM 2009)*, pages 1–6. IEEE, December 2009.

[92] Joseph Lewthwaite and Victoria Smith. Limewire examinations. In *The Proceedings of the Eighth Annual DFRWS Conference*, volume 5, Supplement of *Digital Investigation*, pages S96–S104, September 2008.

[93] Ping Li, Bharadwaj Veeravalli, and Ashraf A. Kassim. Design and Implementation of Parallel Video Encoding Strategies Using Divisible Load Analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(9):1098–1112, September 2005.

[94] Kristian Lumme, Luigia Petre, Petter Sandvik, and Kaisa Sere. Towards Dependable H.264 Video Decoding. In Naveed Ahmed, Daniele Quercia, and Christian D. Jensen, editors, *Workshop Proceedings of the 5th IFIP WG 11.11 International Conference on Trust Management (IFIPTM 2011)*, pages 325–337. Technical University of Denmark, 2011.

[95] Dorian Lynskey. Is Daniel Ek, Spotify founder, going to save the music industry ... or destroy it? `http://www.theguardian.com/technology/2013/nov/10/daniel-ek-spotify-streaming-music`, November 2013. (Accessed October 2015).

[96] Cor Meenderinck, Arnaldo Azevedo, Ben Juurlink, Mauricio Alvarez Mesa, and Alex Ramírez. Parallel Scalability of Video Decoders. *Journal of Signal Processing Systems*, 57(2):173–194, November 2009.

[97] Robert K. Merton. *On the Shoulders of Giants: A Shandean Postscript.* New York: Free Press, 1965.

[98] Mauricio Alvarez Mesa, Alex Ramírez, Arnaldo Azevedo, Cor Meenderinck, Ben Juurlink, and Mateo Valero. Scalability of Macroblock-level Parallelism for H.264 Decoding. In *2009 15th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 236–243. IEEE, December 2009.

[99] Modularisation Plug-in – Event-B. `http://wiki.event-b.org/index.php/Modularisation_Plug-in`. (Accessed October 2015).

[100] Jack Moffitt. Ogg Vorbis – Open, Free Audio – Set Your Media Free. *Linux Journal*, 2001(81es), January 2001.

[101] Ricky K. P. Mok, Edmond W. W. Chan, and Rocky K. C. Chang. Measuring the Quality of Experience of HTTP Video Streaming. In *2011 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 485–492. IEEE, May 2011.

[102] Jacob Jan David Mol, Arno Bakker, Johan A. Pouwelse, Dick H. J. Epema, and Henk J. Sips. The Design and Deployment of a BitTorrent Live Video Streaming Solution. In *11th IEEE International Symposium on Multimedia (ISM '09)*, pages 342–349. IEEE, December 2009.

[103] Jacob Jan David Mol, Johan A. Pouwelse, Michel Meulpolder, Dick H. J. Epema, and Henk J. Sips. Give-to-Get: Free-riding-resilient Video-on-Demand in P2P Systems. In Reza Rejaie and Roger Zimmermann, editors, *Multimedia Computing and Networking 2008*, volume 6818 of *Proceedings of SPIE*, January 2008.

[104] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How Amazon Web Services Uses Formal Methods. *Communications of the ACM*, 58(4):57–73, April 2015.

[105] Sergey Ostroumov, Leonidas Tsiopoulos, Juha Plosila, and Kaisa Sere. Formal approach to agent-based dynamic reconfiguration in Networks-On-Chip. *Journal of Systems Architecture*, 59(9):709–728, October 2013.

[106] K. Nadim Parvez, Carey Williamson, Anirban Mahanti, and Niklas Carlsson. Analysis of BitTorrent-like Protocols for On-Demand Stored Media Streaming. In *SIGMETRICS '08. Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 301–312, June 2008.

[107] Luigia Petre and Petter Sandvik. Formal Modelling of Inter-Peer Relations in Peer-to-Peer Media Distribution Systems. In Paul Pettersson and Cristina Seceleanu, editors, *Proceedings of the 23rd Nordic Workshop on Programming Theory*, Technical Report 254/2011, pages 21—23. Mälardalen Real-Time Research Centre, 2011.

[108] Luigia Petre, Petter Sandvik, and Kaisa Sere. A Modular Approach to Formal Modelling of Peer-to-Peer Networks. Technical Report 1039, Turku Centre for Computer Science, 2012. `http://tucs.fi/publications/view/?pub_id=tPeSaSe12a`.

[109] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in BitTorrent? In *Proceedings of the 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI'07)*, pages 1–14, 2007.

[110] Ken C. Pohlmann. *The Compact Disc Handbook, 2nd ed.* A-R Editions, Inc., 1992.

[111] The ProB Animator and Modelchecker. `http://stups.hhu.de/ProB/`. (Accessed October 2015).

[112] Dongyu Qiu and R. Srikant. Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks. In *Proceedings of the 2004*

conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '04), pages 367–378. ACM, October 2004.

[113] Albert Rafetseder, Florian Metzger, David Stezenbach, and Kurt Tutschku. Exploring YouTube's Content Distribution Network Through Distributed Application-Layer Measurements: A First View. In *Proceedings of the 2011 International Workshop on Modeling, Analysis, and Control of Complex Networks (Cnet '11)*, pages 31–36, 2011.

[114] Matei Ripeanu, Ian Foster, and Adriana Iamnitchi. Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design. *IEEE Internet Computing Journal (special issue on peer-to-peer networking)*, 6(1), 2002.

[115] RODIN – Rigorous Open Development Environment for Complex Systems. `http://rodin.cs.ncl.ac.uk`. (Accessed October 2015).

[116] Kashif Saghar, William Henderson, David Kendall, and Ahmed Bouridane. Formal modelling of a robust Wireless Sensor Network routing protocol. In *2010 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 281–288. IEEE, June 2010.

[117] Asieh Salehi Fathabadi, Colin Snook, and Michael Butler. Applying an Integrated Modelling Process to Run-time Management of Many-Core Systems. In Elvira Albert and Emil Sekerinski, editors, *11th International Conference on Integrated Formal Methods (iFM 2014)*, volume 8739 of *Lecture Notes in Computer Science*, pages 120–135. Springer International Publishing Switzerland, September 2014.

[118] Petter Sandvik. Adapting Peer-to-Peer File Sharing Technology for On-Demand Media Streaming. Master's thesis, Åbo Akademi University, May 2008.

[119] Petter Sandvik. SPECTA: A Formal Specification Language for Content Transfer Algorithms. In Uwe Wolter and Yngve Lamo, editors, *24th Nordic Workshop on Programming Theory*, Reports in Informatics 403, pages 81–83. University of Bergen, 2012.

[120] Petter Sandvik and Mats Neovius. The Distance-Availability Weighted Piece Selection Method for BitTorrent: A BitTorrent Piece Selection Method for On-Demand Streaming. In Antonio Liotta, Nick Antonopoulos, George Exarchakos, and Takahiro Hara, editors, *Proceedings of The First International Conference on Advances in P2P Systems (AP2PS 2009)*, pages 198–202. IEEE Computer Society, 2009.

[121] Petter Sandvik and Mats Neovius. A Further Look at the Distance-Availability Weighted Piece Selection Method: A BitTorrent Piece Selection Method for On-Demand Media Streaming. *International Journal on Advances in Networks and Services*, 3(3 & 4):473–483, 2010. ISSN 1942-2644, `http://www.iariajournals.org/networks_and_services/`.

[122] Petter Sandvik, Kaisa Sere, and Marina Waldén. An Event-B Model for On-Demand Streaming. Technical Report 994, Turku Centre for Computer Science, 2010. `http://tucs.fi/publications/view/?pub_id=tSaSeWa10a`.

[123] Petter Sandvik, Kaisa Sere, and Marina Waldén. Modelling BitTorrent-Like Streaming Piece Selection with Event-B. In Marina Waldén and Luigia Petre, editors, *Proceedings of the 22nd Nordic Workshop on Programming Theory*, TUCS General Publication 57, pages 82–84. Turku Centre for Computer Science, 2010.

[124] Stefan Sariou, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy. An Analysis of Internet Content Delivery Systems. *SIGOPS Oper. Syst. Rev.*, 36(SI):315–327, December 2002.

[125] Rüdiger Schollmeier. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In *Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01)*. IEEE, 2001.

[126] John Seabrook. Streaming Dreams: YouTube Turns Pro. *The New Yorker*, January 16, 2012.

[127] Adrian Segall. Distributed Network Protocols. *IEEE Transactions on Information Theory*, IT-29(1):23–35, January 1983.

[128] Florian H. Seitner, Michael Bleyer, Margrit Gelautz, and Ralf M. Beuschel. Evaluation of data-parallel H.264 decoding approaches for strongly resource-restricted architectures. *Multimedia Tools and Applications*, 53(2):431–457, 2011.

[129] Purvi Shah and Jehan-François Pâris. Peer-to-Peer Multimedia Streaming Using BitTorrent. In *IEEE International Performance, Computing, and Communications Conference (IPCCC 2007)*, pages 340–347. IEEE, April 2007.

[130] Iraj Sodagar. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE Multimedia*, 18(4):62–67, April 2011.

[131] Mike Solomon. Scalability at YouTube. Talk at PyCon 2012, March 9th 2012. https://www.youtube.com/watch?v=G-lGCC4KKok. (Accessed October 2015).

[132] Pyda Srisuresh and Matt Holdrege. IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663. https://tools.ietf.org/html/rfc2663, August 1999. (Accessed October 2015).

[133] Thomas Stockhammer. Dynamic Adaptive Streaming over HTTP – Design Principles and Standards. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSys '11)*, pages 133–144. ACM, February 2011.

[134] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, December 2012.

[135] Gary J. Sullivan, Pankaj Topiwala, and Ajay Luthra. The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions. In Andrew G. Tescher, editor, *Applications of Digital Image Processing XXVII, 454*, volume 5558 of *Proc. SPIE*, November 2004.

[136] Shuwei Sun, Dong Wang, and Shuming Chen. A Highly Efficient Parallel Algorithm for H.264 Encoder Based on Macro-Block Region Partition. In Ronald Perrott, Barbara M. Chapman, Jaspal Subhlok, Rodrigo Fernandes de Mello, and Laurence T. Yang, editors, *High Performance Computing and Communications. Third International Conference, HPCC 2007, Houston, USA, September 26-28, 2007. Proceedings*, volume 4782 of *Lecture Notes in Computer Science*, pages 577–585, September 2007.

[137] Koichi Takagi, Satoshi Miyaji, Shigeyuki Sakazawa, and Yasuhiro Takishima. Conversion of MP3 to AAC in the Compressed Domain. In *IEEE 8th Workshop on Multimedia Signal Processing*, pages 132–135. IEEE, October 2006.

[138] Yap-Peng Tan and Haiwei Sun. Fast motion re-estimation for arbitrary downsizing video transcoding using H.264/AVC standard. *IEEE Transactions on Consumer Electronics*, 50(3):887–894, August 2004.

[139] Saurabh Tewari and Leonard Kleinrock. Analytical Model for BitTorrent-based Live Video Streaming. In *4th IEEE Consumer Communications and Networking Conference (CCNC 2007)*, pages 976–980. IEEE, January 2007.

[140] Brian Tierney, Ezra Kissel, Martin Swany, and Eric Pouyoul. Efficient Data Transfer Protocols for Big Data. In *IEEE 8th International Conference on E-Science (e-Science 2012)*, pages 1–9. IEEE, October 2012.

[141] Erik B. van der Tol, Egbert G.T. Jaspers, and Rob H. Gelderblom. Mapping of H.264 decoding on a multiprocessor architecture. In Bhaskaran Vasudev, T. Russell Hsing, Andrew G. Tescher, and Touradj Ebrahimi, editors, *Image and Video Communications and Processing 2003*, volume 5022 of *Proceedings of SPIE*, 2003.

[142] Niraj Tolia, Michael Kaminsky, David G. Andersen, and Swapnil Patil. An Architecture for Internet Data Transfer. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI '06)*. USENIX, May 2006.

[143] Ruben Torres, Alessandro Finamore, Jin Ryong Kim, Marco Mellia, Maurizio M. Munafo, and Sanjay Rao. Dissecting Video Server Selection Strategies in the YouTube CDN. In *31st International Conference on Distributed Computing Systems (ICDCS 2011)*, pages 248–257, June 2011.

[144] Ricardo Vice Santos. Spotify: P2P music streaming. Slides from talk at ISEL Tech in Lisbon, May 26th, 2011. `http://www.slideshare.net/ricardovice/spotify-p2p-music-streaming`. (Accessed October 2015).

[145] Aggelos Vlavianos, Marios Iliofotou, and Michalis Faloutsos. BiToS: Enhancing BitTorrent for Supporting Streaming Applications. In *9th IEEE Global Internet Symposium 2006*, April 2006.

[146] Marina Waldén and Kaisa Sere. Reasoning About Action Systems Using the B-Method. *Formal Methods in Systems Design*, 13:5–35, 1998.

[147] Patricia Wooster. *YouTube Founders Steve Chen, Chad Hurley, and Jawed Karim*. Stem Trailblazer Bios. Lerner Publications, 2014.

[148] Dongyan Xu, Mohamed Hefeeda, Susanne Hambrusch, and Bharat Bhargava. On Peer-to-Peer Media Streaming. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*. IEEE, 2002.

[149] Dongyan Xu, Sunil Suresh Kulkarni, Catherine Rosenberg, and Heung-Keung Chai. Analysis of a CDN–P2P hybrid architecture for cost-effective streaming media distribution. *Multimedia Systems*, 11(4):383–399, April 2006.

[150] Thomas Canhao Xu, Alexander Wei Yin, Pasi Liljeberg, and Hannu Tenhunen. A Study of 3D Network-on-Chip Design for Data Parallel H.264 Coding. In *NORCHIP 2009*, pages 1–6. IEEE, November 2009.

[151] Lu Yan. A Formal Architectural Model for Peer-to-Peer Systems. In Xuemin Shen, Heather Yu, John Buford, and Mursalin Akon, editors, *Handbook of Peer-to-Peer Networking*, pages 1295–1314. Springer US, 2010.

[152] Nelson H.C. Yung and K.C. Chu. Fast and Parallel Video Encoding by Workload Balancing. In *1998 IEEE International Conference on Systems, Man, and Cybernetics*, volume 5, pages 4642–4647. IEEE, October 1998.

[153] Pamela Zave. Using Lightweight Modeling To Understand Chord. *ACM SIGCOMM Computer Communication Review*, 42(2), April 2012.

[154] Niels Zeilemaker, Mihai Capotă, Arno Bakker, and Johan Pouwelse. Tribler: P2P Media Search and Sharing. In *Proceedings of the 19th ACM international conference on Multimedia (MM '11)*, pages 739–742, November 2011.

[155] Xianrong Zhang and Zhilin Feng. ZJCShare: A Secure Cross-platform P2P Instant Messaging Application Based on De-centralized Structure Model. In *International Conference on Computer Science and Service System (CSSS 2011)*, pages 2846–2849. IEEE, June 2011.

[156] Dajiang Zhou, Peilin Liu, Ji Kong, Yunfei Zhang, Bin He, and Ning Deng. An SoC Based HW/SW Co-Design Architecture for Multi-Standard Audio Decoding. In *IEEE Asian Solid-State Circuits Conference (ASSCC '07)*, pages 201–203. IEEE, November 2007.

Complete List of Publications

1 Petter Sandvik and Mats Neovius. The Distance-Availability Weighted Piece Selection Method for BitTorrent: A BitTorrent Piece Selection Method for On-Demand Streaming. In: Antonio Liotta, Nick Antonopoulos, George Exarchakos, and Takahiro Hara (Eds.), *Proceedings of The First International Conference on Advances in P2P Systems (AP2PS 2009)*, pp. 198–202. IEEE Computer Society, October 2009.

2 Petter Sandvik, Kaisa Sere, and Marina Waldén. Modelling BitTorrent-Like Streaming Piece Selection with Event-B. In: Marina Waldén and Luigia Petre (Eds.), *Proceedings of the 22nd Nordic Workshop on Programming Theory*, General Publication 57, pp. 82–84. TUCS – Turku Centre for Computer Science, November 2010.

3 Petter Sandvik and Mats Neovius. A Further Look at the Distance-Availability Weighted Piece Selection Method: A BitTorrent Piece Selection Method for On-Demand Media Streaming. *International Journal on Advances in Networks and Services*, ISSN 1942-2644, Vol. 3, No. 3 & 4, pp. 473–483. IARIA, 2010
`http://www.iariajournals.org/networks_and_services/`

4 Kristian Lumme, Luigia Petre, Petter Sandvik, and Kaisa Sere. Towards Dependable H.264 Video Decoding. In: Naveed Ahmed, Daniele Quercia, and Christian D. Jensen (Eds.), *Workshop Proceedings of the 5th IFIP WG 11.11 International Conference on Trust Management (IFIPTM 2011)*, pp. 325–337. Technical University of Denmark, June 2011.

5 Luigia Petre and Petter Sandvik. Formal Modelling of Inter-Peer Relations in Peer-to-Peer Media Distribution Systems. In: Paul Pettersson and Cristina Seceleanu (Eds.), *Proceedings of the 23rd Nordic Workshop on Programming Theory*, Technical Report 254/2011, pp. 21–23. Mälardalen Real-Time Research Centre, October 2011.

6 Petter Sandvik and Kaisa Sere. Formal Analysis and Verification of Peer-to-Peer Node Behaviour. In: Antonio Liotta, Nikos Antonopoulos, Giuseppe Di Fatta, Takahiro Hara and Quang Hieu Vu (Eds.), *The Third International Conference on Advances in P2P Systems (AP2PS 2011)*, pp. 47–52. IARIA, November 2011.

7 Luigia Petre, Petter Sandvik, and Kaisa Sere. Node Coordination in Peer-to-Peer Networks. In: Marjan Sirjani (Ed.), *COORDINATION 2012*, Lecture Notes in Computer Science Vol. 7274, pp. 196–211. Springer-Verlag GmbH Berlin Heidelberg, June 2012.

8 Petter Sandvik. SPECTA: A Formal Specification Language for Content Transfer Algorithms. In: Uwe Wolter and Yngve Lamo (Eds.), *24th Nordic Workshop on Programming Theory*, Reports in Informatics 403, pp. 81–83. University of Bergen, October 2012.

9 Kristian Lumme, Luigia Petre, Petter Sandvik, and Kaisa Sere. A Formal Approach to H.264 Video Decoding on Multicore Systems. *Int. J. Critical Computer-Based Systems*, Vol. 4, No. 1, pp. 3–26. Inderscience Publishers, 2013.
`http://dx.doi.org/10.1504/IJCCBS.2013.053740`

10 Maryam Kamali, Mats Neovius, Luigia Petre, and Petter Sandvik. Formal Development of System of Systems. *ISRN Software Engineering 2013*, pp. 1–10. Hindawi Publishing Corporation, 2013.

11 Seppo Horsmanheimo, Maryam Kamali, Mikko Kolehmainen, Mats Neovius, Luigia Petre, Mauno Rönkkö, and Petter Sandvik. On Proving Recoverability of Smart Electrical Grids. In *NASA Formal Methods: 6th International Syposium (NFM 2014)*, Lecture Notes in Computer Science Vol. 8430, pp. 77–92. Springer-Verlag, April 2014.

12 Petter Sandvik. SPECTA: A Formal Specification Language for Content Transfer Algorithms. In *IEEE Fifteenth International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2014)*, pp. 638–641. IEEE, June 2014.

Part II

Original Publications

Paper I

A Further Look at the Distance-Availability Weighted Piece Selection Method: A BitTorrent Piece Selection Method for On-Demand Media Streaming

Petter Sandvik and Mats Neovius

Originally published in *International Journal on Advances in Networks and Services*, ISSN 1942-2644, Vol. 3, No. 3 & 4, pp. 473–483. IARIA, 2010
`http://www.iariajournals.org/networks_and_services/`

Based on the publication: Petter Sandvik and Mats Neovius. The Distance-Availability Weighted Piece Selection Method for BitTorrent: A BitTorrent Piece Selection Method for On-Demand Streaming. In: Antonio Liotta, Nick Antonopoulos, George Exarchakos, and Takahiro Hara (Eds.), *Proceedings of The First International Conference on Advances in P2P Systems (AP2PS 2009)*, pp. 198–202. IEEE Computer Society, October 2009.

A Further Look at the Distance-Availability Weighted Piece Selection Method

A BitTorrent Piece Selection Method for On-Demand Media Streaming

Petter Sandvik and Mats Neovius
Department of Information Technologies
Åbo Akademi University
and
Turku Centre for Computer Science
Turku, Finland
e-mail: {petter.sandvik, mats.neovius}@abo.fi

***Abstract*—During the last few years, BitTorrent has become a popular way of transferring large files over the Internet. However, the original out-of-order nature of the BitTorrent protocol has made it difficult to enable playback of media files that have not yet been fully transferred. In this paper we describe a piece selection method which we believe will enable simultaneous playback of the transferred media file without impacting on the speed and quality of the transfer. The distance-availability weighted method compromises between selecting rare pieces and pieces which are soon to be played back, making playback possible before the transfer is complete. In our simulations, we have compared our piece selection method with other proposals for on-demand streaming media using a BitTorrent-like setup, with our method giving similar or better results.**

Keywords-media, on-demand, peer-to-peer, streaming, BitTorrent, simulation

I. INTRODUCTION

In [1], we presented the distance-availability weighted method; a BitTorrent piece selection method for on-demand streaming. BitTorrent is originally a peer-to-peer file sharing protocol and an application, designed by Bram Cohen and first released in July 2001 [2]. During the following years BitTorrent evolved into one of the most popular peer-to-peer protocols [3][4]. Unlike earlier popular peer-to-peer file sharing applications such as Napster and Kazaa, the use of BitTorrent usually starts by clicking a link in a web browser, and Cohen suggests that "ease of use has contributed greatly to BitTorrent's adoption, and may even be more important than [...] the performance and cost redistribution features" [5]. Another major difference between the earlier peer-to-peer file sharing applications and BitTorrent is the lack of central server in the latter, in the sense that BitTorrent users are not all connected to each other through one server.

While BitTorrent is popular, there are also other reasons for choosing it as the basis for a peer-to-peer based media streaming system such as the one we have in mind. Several applications using the protocol, as well as the protocol itself, are open, which makes understanding and modifying more easily possible than if the staring point was a proprietary product. For our purposes an even more important aspect is that all the logic involved in the file transfer is contained on the client side, which makes it possible, at least in theory, to have an on-demand content streaming BitTorrent client participate in content transfer with regular, non-streaming, BitTorrent clients.

While the original BitTorrent protocol was not designed for streaming, it has already been argued [6] that with some modifications, it would be possible to create a streaming media solution based on BitTorrent. Indeed, there are proprietary and commercial efforts to create exactly such a thing, for instance "to turn BitTorrent into a point-click-watch experience much more similar to YouTube" [7], and in [1] we described our proposal for a solution that would enable file sharing in such a way that content could be played back while downloading. In this paper we will further describe our proposal and show that it is a modification to the BitTorrent protocol, which would allow it to function as an on-demand media streaming solution.

The rest of this paper is organised as follows: in Section II, we describe BitTorrent and its terminology, as used throughout this paper. Section III consists of the requirements of an on-demand streaming media application in general, and what assumptions we will make for a BitTorrent streaming application to be feasible. In Section IV we present our proposed piece selection method, and in Section V, we compare it to existing piece selection methods that could be used for on-demand streaming. In Section VI, we list a selection of related works. The paper is concluded in section VII with a discussion about our findings and possible future work on this subject.

II. BITTORRENT

Since BitTorrent was introduced by Bram Cohen in 2001 it has evolved. While the terminology used in [5] could be seen as a standard, the terminology used in this paper will be based on that of [8]. This terminology is close to that of the application Vuze, formerly known as Azureus, a BitTorrent client often used as a basis for research applications [9][10][11].

A. BitTorrent Terminology

- **Torrent**. A torrent consists of a single file, or a collection of files, to be shared, and the associated metadata. The metadata, and therefore the torrent itself, is uniquely identified by its info-hash [12].

- **Tracker**. A tracker is a piece of software that is involved in keeping track of which peers are involved in the transfer of a particular torrent, using the info-hash of the torrent. Each torrent can be associated with many trackers. Additions to the BitTorrent protocol have enabled peer discovery through other means, such as distributed hash tables and peer exchange, and thus the use of trackers is no longer required. Whether a tracker is used or not does not affect the file transfer, and this is of little importance to us.
- **Pieces** and **blocks**. The data of a torrent is divided into pieces, and each piece is divided into blocks, also called sub-pieces. A piece is typically 256 kilobytes in size [8][13] while a block is typically 16 kilobytes in size [5][8]. A piece must be complete, that is, all blocks of it must have been downloaded, before the piece can be transferred to another peer.
- **Torrent index file**. Also known as a ".torrent" [5], a torrent index file contains information about the torrent, such as the universal resource locator (URL) of the tracker (or trackers, if any), piece size of the torrent, names and sizes of the files in the torrent, as well as SHA-1 hashes of all the pieces. This file is generally hosted on a web server and downloading of a torrent starts by opening the file with a BitTorrent application.
- **Interested** and **to choke**. If peer B has pieces, which peer A does not have, peer A is interested in peer B, otherwise peer A is uninterested in peer B. If peer B decides not to send data to peer A, peer B chokes peer A, and if peer B decides to send data to peer A, peer B unchokes peer A.
- **Peer set** and **active peer set**. A peer set consists of all the other peers one peer is connected to, and the active peer set consists of those peers it is currently sending data to, i.e., its unchoked peers.
- **Seed**. A peer that has all pieces of a torrent and therefore only sends data is called a seed.
- **Availability**. We define availability of a piece as the number of peers in the peer set who have that specific piece.

B. How BitTorrent Data Transfer Works

When transferring data, BitTorrent needs to decide what data to request (piece selection) and which peers to choke or unchoke (peer selection). The reference BitTorrent implementation begins by selecting pieces to download at random, until one complete piece has been downloaded. BitTorrent then switches to a rarest-first piece selection method. The rarest-first selection method selects the piece that the fewest peers have, i.e., the piece with the lowest availability, as the first piece to request. This has the effect of reducing the possibility that one piece may become unavailable, as a lower number of peers having a piece makes other peers more likely to request that particular piece, thereby increasing the number of peers that will have that piece. When a single block from a piece has been downloaded, other blocks from that piece are given highest priority, in order to have as few incomplete pieces transferred as possible. BitTorrent then keeps selecting the rarest pieces first, until all remaining blocks in all remaining pieces have been requested, at which time all remaining blocks are requested from all peers in the active peer set. This is done so that one slow peer cannot prevent the whole download from completing.

To create an incentive for peers to upload as well as download, the reference BitTorrent implementation uses a tit-for-tat (TFT) peer selection strategy. Every ten seconds, which peers to unchoke is evaluated based on the rate of data sent, with the fastest peers chosen as the peers to unchoke. Additionally, there is also one interested peer unchoked at random, re-evaluated every thirty seconds. This randomly chosen peer is called the optimistic unchoke [5][8] and exists for two reasons: to allow new peers a chance to enter the TFT game, and to potentially discover faster peers that could become regular, non-optimistic, unchokes [8][9]. This approach is not necessarily the optimal way of peer selection, and alternative approaches have been suggested, such as [9]. However, the basic TFT strategy remains an essential part of the BitTorrent protocol as used today.

After a download is finished, the peer may continue to participate in sending data to other peers, and in many cases the peer is actually encouraged to do so. In this case choosing peers based on how much they send is of course not possible, and thus the reference BitTorrent implementation then switches to sending to the peers that can receive data the fastest [5]. However, this feature could be exploited, and later clients have switched to choosing peers to send to randomly [9].

III. REQUIREMENTS FOR STREAMING

We define streaming as the transport of data in a continuous flow, in which the data can be used before it has been received in its entirety. In this context, on-demand streaming is essentially playback, as a stream, of pre-recorded content, at the request of a user. This is in contrast to live streaming, which is playback, as a stream, of content, which is not pre-recorded but rather created and transmitted practically simultaneously. Concerning an on-demand peer-to-peer media streaming system, we make the following initial observations:

Each peer must have a download bandwidth at least as large as the playback bit rate of the media. Unlike a peer-to-peer file sharing system, the media is played while being received, and therefore cannot be received slower than it should be played back. Furthermore, if the media is to be received faster than real-time, it must also be sent faster than real-time. Therefore, the average upload bandwidth of all peers must be larger than the playback bit rate of the media, although an individual peer may have an upload bandwidth smaller than that.

We will make the following assumptions regarding a BitTorrent-based on-demand media streaming system: The torrent will consist of only one complete media file, unlike in file sharing where several files can be combined in one torrent. Additionally, in a file sharing system the time an

individual peer participates is difficult to estimate and not dependent on the content received. In an on-demand peer-to-peer streaming system it can be estimated more easily, and although peers may of course leave the system at any time, our assumption is that a typical peer will enter the system when starting playback and leave the system some time after playback is complete, which means some time after all pieces of the content have been received. We will also assume that for each piece there will always be at least one peer holding it, that is, we assume that we do not encounter the situation where the availability of any piece is zero, because that would lead to a situation where complete playback is not possible.

Internet connections are typically symmetric, with the same bandwidth available for sending and receiving data, or asymmetric with a higher download bandwidth than upload bandwidth. Therefore, a typical peer will be able to receive data at least as fast as it can send data. Additionally, all pieces must be requested, resulting in a complete file once the transfer is complete, unless the peer leaves before finishing playback. Furthermore, each peer will keep and make available all the pieces it has received, for as long as the peer is participating. Each peer must therefore have enough space to store the entire media file.

Ideally, the piece selection algorithm in our BitTorrent-based streaming system should comprise the following behaviours: When the ratio of seeds to peers is high, our piece selection method should prefer pieces close to being played back rather than rare pieces, because with a large number of complete sources there is no need for downloading rare pieces to ensure the future availability of all pieces. In the extreme case, where our client has the only incomplete copy of the content, there is no obvious downside to requesting pieces sequentially. On the other hand, when the ratio of seeds to other peers is low, our piece selection method should also choose rare pieces to improve the overall availability of the pieces, while still requesting enough pieces in sequence to make continuous playback possible.

Although we specified earlier that peers should be able to download faster than the playback rate of the media, there will be variations in how much faster the peers will be able to receive data. Ideally, our piece selection method will be able to adapt to these kinds of differing conditions. For instance, if a peer can download data only slightly faster than the playback rate, our piece selection method should ensure that the peer requests data mostly sequentially, while a peer that can download the media much faster than its playback speed should also frequently request rare pieces in order to improve the overall availability of the pieces. With that in mind, we present our proposal for a piece selection method for on-demand streaming.

IV. The Distance-Availability Weighted Method

The idea behind the distance-availability weighted method for piece selection (DAW) is to strike a balance between lots of consecutive pieces, which is good for playback, and requesting rarest pieces first, which is good for piece availability. In other words, we want to balance requesting between distance (in the sequence of pieces) and availability.

We start by having a small, fixed buffer of size k. The priority for requesting the pieces in the buffer will be the highest, here represented as 1. Outside the buffer we will calculate the priority for each not yet requested piece as

$$\text{Priority} = 1/((P_r - P_c) * m_r)$$

where P_r is the sequence number of a particular piece, m_r is the number of peers who hold that piece, and P_c is the sequence number of the current last piece in the buffer. In other words, $P_r - P_c$ is the distance to a particular piece and m_r is the availability of that piece. The priority for a piece outside the buffer is therefore never more than 1, with a priority of 1 occurring only in the rare situation that the piece immediately outside the buffer is held by exactly one peer. Pieces that are further from being played back or held by more peers are given lower priorities, while a short distance from the buffer or a low availability increases the priority.

The availability of a particular piece and its distance from the last piece in the buffer are here equally weighted when determining the priority. However, we do not claim that giving equal weights to distance and availability is in any way the optimal solution. We have not extensively tested different weights, but it seems likely that factors such as the total number of pieces, the number of peers, and the network speed of the peers, will have an effect on how the distance and availability should be weighted for optimal performance. As it would not be possible to test all different combinations of weights under all circumstances, we choose here to weigh them equally for the sake of simplicity.

It should be noted that when we talk about availability of a piece we do not mean how many peers in total are involved in the torrent and hold that particular piece. What we actually look at is how many peers in one peer's active peer set hold that particular piece. As one peer need not necessarily be connected to all other peers, especially if the total number of peers is very large, we must by necessity look at the system from one peer's point of view. Another point worth mentioning is that the above priority calculation holds even if we allow playback to start somewhere else than from the beginning. Not yet requested pieces earlier in sequence than the last piece of our playback buffer will get negative priorities, and therefore will not be requested until all pieces higher in sequence have been requested.

V. Comparison with Other Piece Selection Methods

We have done two separate sets of simulations. The first set, the results of which also appeared in [1], is less exhaustive and compares our DAW piece selection method to two others:

- **Sequential Method**. This represents how a straightforward streaming would work, by always requesting pieces in the same order as they appear in the torrent.

- **Rarest-First Method with Buffer (RFB)**. This is the original BitTorrent piece selection method, slightly modified to better support streaming media. This is done as follows: we add a small buffer of fixed size, so that *k* pieces after the currently playing one are requested with the highest priority. If all *k* buffer pieces have been requested, we use the rarest-first method on the remaining part of the media. Another small modification is that if more than one piece have the same availability the original rarest-first method chooses between them randomly [12], while we always choose the one closest to being played back, as in the rarest-first method mentioned in [6].

In our second set of simulations, we add another piece selection method to the comparison:

- **BitTorrent Streaming (BiToS)**. This piece selection method is described in [6]. Pieces not yet downloaded are divided between a small high-priority set, with pieces close to being played back, and a larger remaining pieces set, with lower priority. The probability of choosing a piece from the high-priority set is *p* and the probability of choosing a piece from the remaining pieces set is similarly 1-*p*. Within each set, pieces are chosen rarest-first, with the aforementioned modification that if there is more than one piece with equal availability, the one closest to being played back is chosen. In [6], *p* was chosen to be 0.8 and we have used the same number here.

A. Our First Set of Simulations

In these simulations, we have focused on the piece selection algorithms, and the results therefore do not reflect real-world performance of applications. Our main focus has been on two things: the percent of requests going to the original source over time, and the availability of the last piece (which is also the rarest piece, in these cases) over logical time. The first one of these we want to be as low as possible, because a good peer-to-peer system should distribute the load equally over as many peers as possible and therefore not have a proportionally high amount of requests directed to the original source. The second one we want to be high, as we want the availability of the rarest piece to increase, as that signifies redundancy and robustness of the system. In these simulations we also assume that peers have the ability to send data to as many other peers as necessary, effectively creating a situation where a peer will always get the piece it requests.

The number of pieces was chosen to be 1000; large enough to study the behaviour of different piece selection methods over long periods of time. The buffer size for both DAW and RFB was set to 8 pieces; large enough for smooth playback but small enough that filling the buffer should not impact n overall performance. All simulations were done up to 800 logical time units; at the rate of one request per time unit we therefore never reach the situation that any peers have finished downloading, instead focusing on what happens from the start onwards.

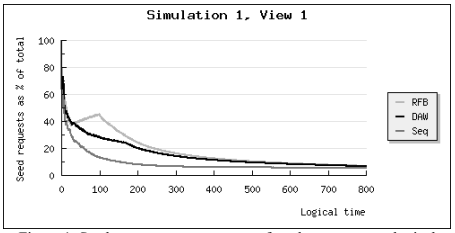


Figure 1. Seed requests as percentage of total requests over logical time, with peers joining at regular intervals.

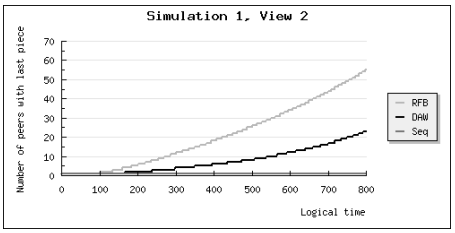


Figure 2. Availability of last piece over logical time, with peers joining at regular intervals.

In the first simulation, to stress the system we chose to have one seed and 100 regular peers, the latter arriving at regular intervals (one new peer every two logical time units). Fig. 1 shows that DAW here results in a lesser burden for the seed than RFB, although it is not as good as the sequential method. However, as Fig. 2 shows, the DAW does not increase the availability of the last piece as much as RFB. With RFB many rare pieces are requested early. These pieces are only available from the seed, and as seen in Fig. 1 the burden on the seed drops only after 100 logical time units. This corresponds to the point where there is no piece left where the seed is the only source, as can be seen in Fig. 2. The same drop can be seen, less dramatically, for DAW around logical time 170, with the same explanation.

The second simulation is identical to the first one, except that all regular peers join simultaneously. As can be seen in Fig. 3, the sequential method does not work in this theoretical situation, while DAW is the better suited one of the other two. Fig. 4 shows the situation as very similar to the one in Fig. 2, except that with more peers from the start it takes less time to increase the availability of the last (rarest) piece with DAW and RFB.

For our third simulation, we chose a similar setup to the first one, but with ten seeds instead of one. Fig. 5 shows the average percentage of requests over logical time to each one of the ten seeds, and DAW is again a less taxing choice than RFB. A possible reason for this can be seen in Fig. 6;

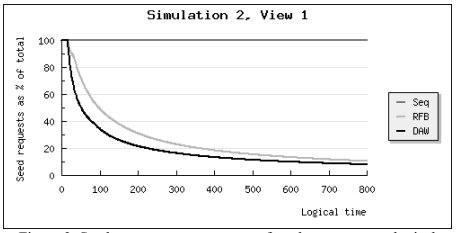


Figure 3. Seed requests as percentage of total requests over logical time, with peers joining simultaneously.

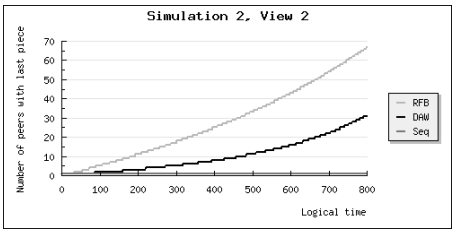


Figure 4. Availability of last piece over logical time, with peers joining simultaneously.

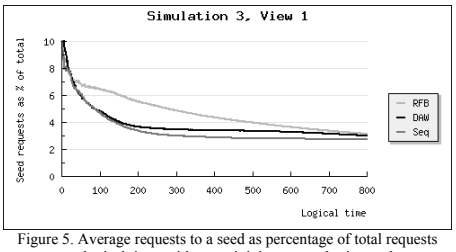


Figure 5. Average requests to a seed as percentage of total requests over logical time, with peers joining at regular intervals.

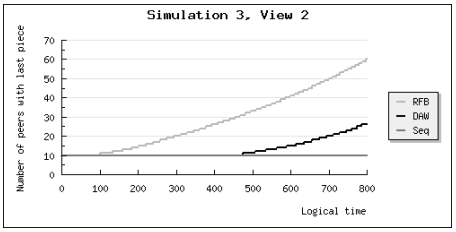


Figure 6. Availability of last piece over logical time, with peers joining at regular intervals.

compared to DAW, RFB spends a lot of time requesting rare pieces although there is no immediate reason for doing so.

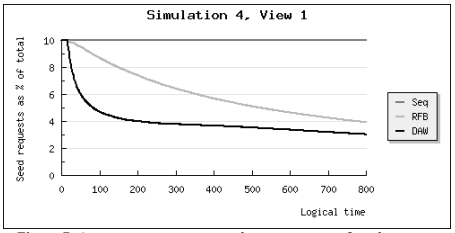


Figure 7. Average requests to a seed as percentage of total requests over logical time, with peers joining simultaneously.

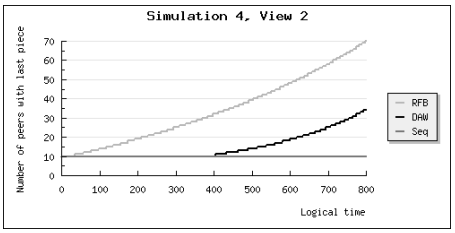


Figure 8. Availability of last piece over logical time, with peers joining simultaneously.

Our fourth simulation is a combination of the second and third ones, with ten seeds and all the regular peers starting at the same time. In Fig. 7 we see that the DAW piece selection method is again less demanding on the seeds than the RFB method, with the sequential method making on average a constant ten percent of the requests to each seed in this theoretical case. Fig. 8 is very much like Fig. 6, showing that the distance-availability weighted method does not spend a lot of time requesting the rarest piece until fairly late.

B. Our Second Set of Simulations

Compared to our first set of simulations, our second set is a step or two closer to reality. Unlike our first set of simulations, where a peer would always get the piece it requested, we have here introduced limits to how many other peers the seeds and regular peers can send to. This introduces the possibility that a peer will not have the piece that it is supposed to be playing back. In our simulations we have dealt with that situation in three different ways:

- **Skip**. We ignore the piece we should be playing back and just note that that piece has been skipped. This is the method favoured by BiToS [6], and it should be noted that for this to work in practise the format of the media content must be tolerant of missing data.
- **Stop**. If we are missing the piece that should be played back, we stop playback until we have been able to receive all pieces in our buffer (or high-priority set), after which we resume playback. From an end user point of view this is similar to how many current on-demand streaming media systems work,

in that not receiving data fast enough means that playback is paused until an amount of consecutive data has been received.

- **Skip and stop**. If we are missing the piece we are supposed to play back we skip it, but if our buffer (or high-priority set) is completely empty we stop playback until we have received all pieces in it. This should in theory generate less complete stops than by always stopping, and possibly also less skips than by just skipping.

What we actually measure in these simulations is three things: firstly, the playback position of the peers after a certain time; secondly, the number of skips and/or stops encountered before that time; and thirdly, the percentage of pieces that should have been transferred that were actually skipped and/or the number of stops per 100 pieces played back, respectively. What we are looking for is thereby a high number for the playback position, but low numbers for the amount of skips and stops.

In all these simulations, we have 10 seeds and 90 regular peers. The seeds can upload to 8 peers simultaneously and the regular peers to 2 peers simultaneously, at the rate of half the playback speed for each peer. The regular peers request new pieces twice as fast as the playback speed. In all the following figures we look at the situation after a logical time of 800. In simulations 5, 6 and 7 the regular peers join simultaneously, while in simulations 8, 9 and 10 they join at regular intervals; similarly to simulations 1 and 3. All simulations were run multiple times and the maximum reported here is the maximum for any peer during any run, the minimum reported is the minimum for any one peer duing any run, and the average reported is the average for all peers over all runs.

Our simulation number five concerns the situation where any pieces not transferred are skipped completely. Fig. 9 shows the maximum, average and minimum playback positions of peers using the BiToS, DAW, RFB and sequential piece selection methods. BiToS seems worse than the others in this case, but it must be pointed out that in BiToS we always spend about 20% of our time downloading pieces not close to being played back, and therefore it is reasonable to expect it to take longer for playback to start. In a best-case scenario this would lead to a lower number of pieces skipped later on, but as we see in Fig. 10, there is not a significant difference in the absolute number of skipped pieces, and if we look at the relative numbers, i.e., the percentage of pieces that should have been played back that were skipped, as in Fig. 11, we find that arguably BiToS is worse than the other three, although the differences are small.

Our simulation number six concerns the situation where the piece to be played back not being available leads to a complete stop of the playback. Fig. 12 shows that the distance-availability weighted method here leads to the furthest playback position. In Fig. 13 we see that despite the good results for the playback position, DAW also does pretty well in the absolute number of stops by coming in second. Fig. 14 shows the relative number of stops, i.e., the number

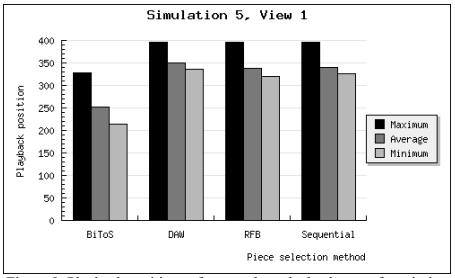


Figure 9. Playback positions of peers when playback stops for missing pieces, with peers joining simultaneously.

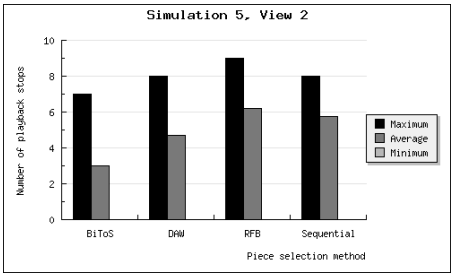


Figure 10. Number of playback stops for each peer when playback stops for missing pieces, with peers joining simultaneously.

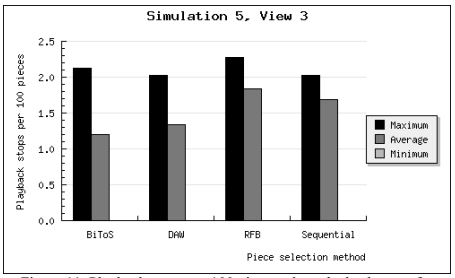


Figure 11. Playback stops per 100 pieces when playback stops for missing pieces, with peers joining simultaneously.

of stops per 100 pieces played back, and there is not a big difference between the four piece selection methods.

Simulation number seven seems to be the one with the most diverse results so far. Here we have the situation that if the piece to be played back is not received, we skip it, but if we do not have any of the pieces in our buffer or high-priority set, we stop. Fig. 15 shows the playback positions of the peers, and the situation is not very different from in the two preceding simulations, with DAW slightly ahead of the others and BiToS slightly behind. However, in Fig. 16 we notice that BiToS seems to skip a lot more than the others, and in Fig. 17 we notice that BiToS stops a lot less often.

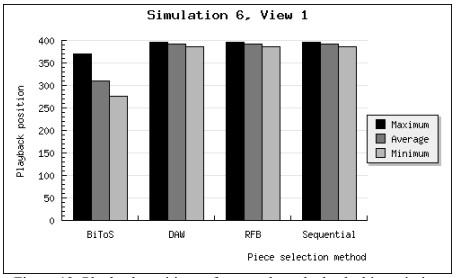


Figure 12. Playback positions of peers when playback skips missing pieces, with peers joining simultaneously.

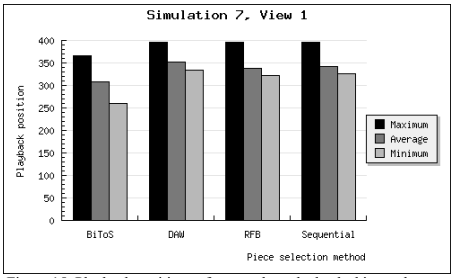


Figure 15. Playback positions of peers when playback skips and stops, with peers joining simultaneously.

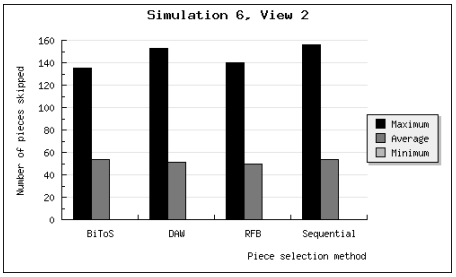


Figure 13. Number of pieces skipped for each peer when playback skips missing pieces, with peers joining simultaneously.

Figure 16. Number of pieces skipped for each peer when playback skips and stops, with peers joining simultaneously.

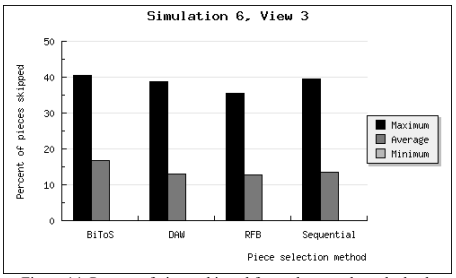


Figure 14. Percent of pieces skipped for each peer when playback skips missing pieces, with peers joining simultaneously.

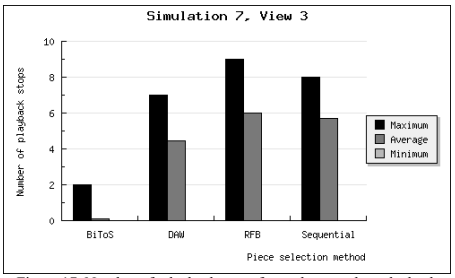


Figure 17. Number of playback stops for each peer when playback skips and stops, with peers joining simultaneously.

This can be explained by how these piece selection methods work. BiToS always requests pieces in a rarest-first, out-of-order fashion, and therefore it is likely that even if the piece to be played back is missing, the high-priority set is not empty, and therefore we just skip. DAW and RFB as described in this paper both have a small buffer in which pieces are requested in-order, and therefore it is likely that if the piece to be played back is missing, the whole buffer is empty, and therefore we stop. In the case of the sequential method, we always request sequentially, and therefore if the piece to be played back is missing, the following k pieces are also missing and we stop.

The following three simulations are similar to the previous ones, except that the regular peers do not join simultaneously. Figures 18, 19 and 20 show the result of simulation 8, where playback stops if the piece to be played back is not available, and the regular peers join at intervals. Because peers do not join at once, the difference between the maximum playback position and the minimum playback position is larger than in simulation 5 (compare Figures 9 and 18). Fig. 19 shows the absolute number of playback stops and Fig. 20 the number of playback stops per 100 pieces, and we see that the DAW and sequential piece selection methods are almost equal in this case, with both

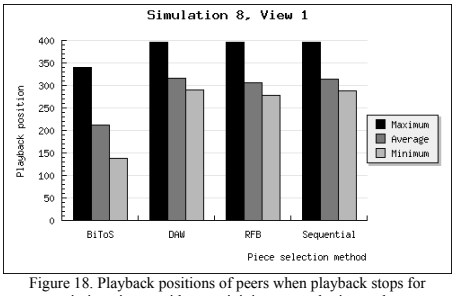


Figure 18. Playback positions of peers when playback stops for missing pieces, with peers joining at regular intervals.

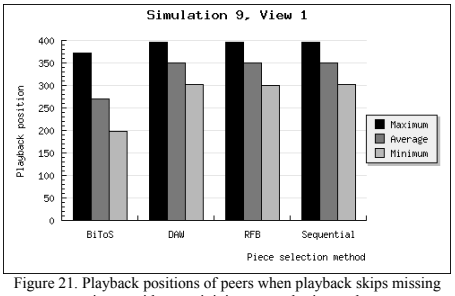


Figure 21. Playback positions of peers when playback skips missing pieces, with peers joining at regular intervals.

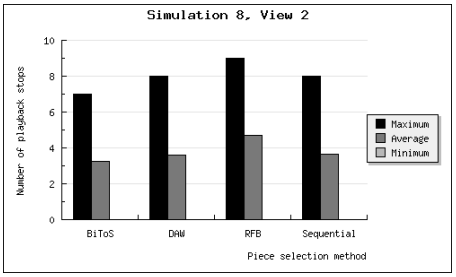


Figure 19. Number of playback stops for each peer when playback stops for missing pieces, with peers joining at regular intervals.

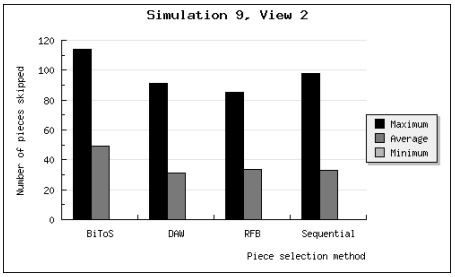


Figure 22. Number of pieces skipped for each peer when playback skips missing pieces, with peers joining at regular intervals.

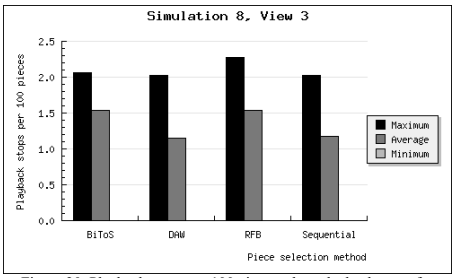


Figure 20. Playback stops per 100 pieces when playback stops for missing pieces, with peers joining at regular intervals.

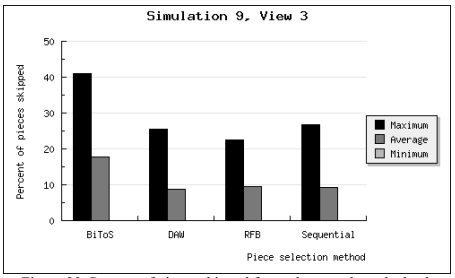


Figure 23. Percent of pieces skipped for each peer when playback skips missing pieces, with peers joining at regular intervals.

being better than the RFB method and in the relative case also better on average than BiToS.

Fig. 21 is comparable to Fig. 12, and again the larger difference between the maximum and minimum playback positions is caused by peers joining at intervals, instead of simultaneously. Comparing Fig. 22 and Fig. 13 we notice that not having all peers joining at once improves the result for the piece selection method utilising some form of sequential requests, leaving BiToS behind. This is emphasised in Fig. 23 where we see the percent of skipped pieces instead of the absolute amount.

Simulation 10 is comparable to simulation 7, and comparing Fig. 24 with Fig. 15 shows that also in this case the peers joining at intervals has the effect of putting BiToS further behind when it comes to playback position. Fig. 25 and Fig. 26 show that as in simulation 7, the nature of BiToS makes it skip more often than stop, while the sequential buffer used in the DAW and RFB piece selection methods make them behave more like the sequential method.

So far, we have only compared the piece selection methods to each other but not discussed the actual figures. While this is of course a very theoretical simulation, the bandwidth figures we have used suggest that playback

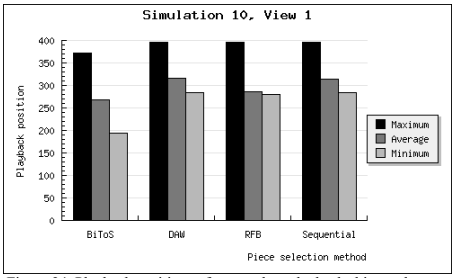


Figure 24. Playback positions of peers when playback skips and stops, with peers joining at regular intervals.

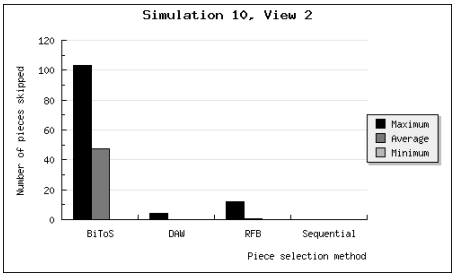


Figure 25. Number of pieces skipped for each peer when playback skips and stops, with peers joining at regular intervals.

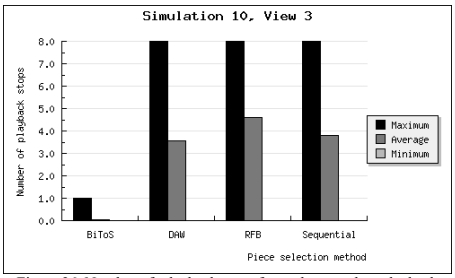


Figure 26. Number of playback stops for each peer when playback skips and stops, with peers joining at regular intervals.

should theoretically be possible for all peers without interruption. Instead, we get figures up to above 40% of all pieces skipped (for instance in Fig. 10) with average levels of more than 1/8 of all pieces skipped (for instance in Fig. 10 and 16). The situation for stops is not very good either, with an average of at least one stop for every 100 pieces played back (Fig. 14). Neither of these results is very good from an end-user point of view. The conclusion we can draw from this is that we need a better way of determining when to start, or restart, playback, as the methods we have used here do not seem to work in that regard. As for comparing the performance of DAW to the others, we note that in there three simulations DAW always comes up as the method that gives the best results with regard to playback position, without ever being the worst in any other regard.

One major concern regarding piece selection methods is whether they work well with the peer selection methods, in other words, whether the method of selecting pieces to request is compatible in practise with the methods used to determine which peers to send data to. The original tit-for-tat method from BitTorrent requires that data is exchanged both ways between peers in order to function well, and therefore is not a good solution when combined with the sequential method, where data is received from peers with more pieces and sent to peers with fewer pieces, exclusively. RFB would obviously work rather well in this fashion also, and the division of pieces into a high-priority set and a remaining pieces set as in BiToS is because the acquisition of pieces from the latter is "beneficial due to the Tit-for-Tat policy" [6]. We have not done any testing of the distance-availability weighted method on this subject, but our estimate is that it would be compatible. When the ratio of seeds to regular peers is low, the availability of pieces is important and the distance-availability weighted method selects pieces in a manner reminiscent of the rarest-first method, where the tit-for-tat policy has been proved as working. Conversely, when the ratio of seeds to regular peers is high, the distance-availability weighted method behaves similarly to the sequential method, but as the seeds do not require data to be sent to them, the need for out-of-order transfers in order to get the tit-for-tat policy working diminishes. We therefore believe our method would work in such a case as well.

VI. RELATED WORK

This paper is an extended version of [1], which is a further development of a concept introduced in [14]. Whether due to its popularity or some other factor, such as its lack proprietarity, BitTorrent has been the subject of several other research projects during the last few years; some of which are directly relevant to ours.

The BitTorrent protocol has been analysed in real-world usage and found to be an efficient and viable solution for file sharing [8][15]. When it comes to the idea of on-demand streaming with BitTorrent as a basis, one of the more interesting propositions is the previously mentioned BiToS [6]. The main difference between BiToS and our distance-availability weighted method is that BiToS seems designed to maximise the amount of received pieces in a situation where the media bit rate is the same as the download rate of the client, while our solution is designed for a situation where the download bit rate is higher than the media bit rate, and all pieces should be received is such a way that playback is possible before all of the file has been transferred.

The approach used in BiToS is further expanded in [16], which also adds modifications to how the peers choose which peers to send data to, effectively replacing the tit-for-tat peer selection method used in BitTorrent with a method called Give-to-Get. The application Tribler uses Give-to-Get [17], while its protocol also contains other additions such as social networking. Another project combining social networking with BitTorrent and media streaming is

OneSwarm [10], adding "friend-to-friend" file sharing and, as of version 0.6 and later, the ability to choose between "streaming" media files (downloading sequentially) or not (downloading using rarest-first) [18]. As mentioned in Section I, there is also a project underway to "turn BitTorrent into a point-click-watch experience much more similar to YouTube", based on the popular, closed source BitTorrent application µTorrent [7].

Another BitTorrent-based service is LiveBT [19], which replaces the rarest-first method with Most-wanted-Block-Download-First (MBDF). In MBDF, a peer has a set of most wanted blocks (pieces), which is a fixed number of undownloaded pieces. Each peer also knows the most wanted blocks of its peers. With a probability *p* the peer's own most wanted piece is selected, and with probability 1-*p* the most wanted piece of its peers.

The idea of using a weighted priority is not unique to the DAW method. Wu et al [20] propose a weighted piece selection method, but for downloading instead of streaming. Their idea is that peers which hold many pieces are given greater weight than peers with few pieces when computing priorities. Whether this could improve performance also in streaming remains to be seen.

Besides BitTorrent-based solutions, there are also other projects underway to use peer-to-peer networks for on-demand streaming. Peer-to-peer networks serve as the backbone of both Spotify [21] and Voddler [22], the former a platform for on-demand streaming music, while the latter enables on-demand movies. Both these services distribute commercial content and therefore have limitations on usage as well as do not provide technical details on how their networks function.

VII. Conclusion and Future Work

In our first set of simulations, the distance-availability weighted piece selection method seems to be a better solution for on-demand streaming than either a straightforward sequential method or a modified version of the rarest-first method. Our second set of simulations, where in addition to the previously mentioned piece selection methods we also include the one presented in [6], do not show results contradicting the first set of simulations. However, the differences seem to be smaller than we previously thought, and none of the piece selection methods simulated seemed to be sufficiently good to work perfectly in the conditions given. However, as we have still only simulated the theoretical performance of the piece selection methods, we cannot comment on how they would work in a real-life environment. Future work on the subject could include practical implementation and real-world testing of the distance-availability weighted piece selection method, as well as comparison to other piece selection methods for streaming than the ones used for comparison here.

Acknowledgement

We thank Professor Kaisa Sere who has been very helpful during this project.

References

[1] P. Sandvik and M. Neovius, "The Distance-Availability Weighted Piece Selection Method for BitTorrent: A BitTorrent Piece Selection Method for On-Demand Streaming", In Proceedings of AP2PS '09, Sliema, Malta, October 2009

[2] B. Cohen, "BitTorrent - a new P2P app", Yahoo eGroups, http://finance.groups.yahoo.com/group/decentralization/message/3160 (Accessed August 2010)

[3] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy and M. Faloutsos, "File-sharing in the Internet: A characterization of P2P traffic in the backbone", Technical report, November 2003.

[4] Ipoque Internet Study 2008 / 2009. Available from http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009 (Accessed August 2010)

[5] B. Cohen, "Incentives Build Robustness in BitTorrent", In Proc. of IPTPS, 2003.

[6] A. Vlavianos, M. Iliofotou and M. Faloutsos, "BiToS: Enhancing BitTorrent for Supporting Streaming Applications", 9th IEEE Global Internet Symposium 2006 (in Conjunction with IEEE INFOCOM 2006).

[7] µTorrent Labs: Project Falcon, http://www.utorrent.com/labs/falcon (Accessed August 2010)

[8] A. Legout, G. Urvoy-Keller and P. Michiard, "Rarest First and Choke Algorithms Are Enough", In Proceedings of ACM SIGCOMM/USENIX IMC'2006, Rio de Janeiro, Brazil, October 2006.

[9] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy and A. Venkataramani, "Do incentives build robustness in BitTorrent?", 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 2007).

[10] T. Isdal, M. Piatek, A. Krishnamurthy and T. Anderson, "Friend-to-friend data sharing with OneSwarm", Technical report, UW-CSE, February 2009.

[11] D. Choffnes and D. Bustamante, "Taming the Torrent: A practical approach to reducing cross-ISP traffic in P2P systems", In Proceedings of ACM SIGCOMM 2008, August 2008.

[12] A. Nordberg, "Introduction to BitTorrent", Umeå University, 2006. Available from http://www.rasterbar.com/products/libtorrent/bittorrent.pdf (Accessed August 2010)

[13] B. Cohen, "The BitTorrent Protocol Specification", January 2008, http://www.bittorrent.org/beps/bep_0003.html (Accessed August 2010)

[14] P. Sandvik, "Adapting Peer-to-Peer File Sharing for On-Demand Media Streaming", Master of Science Thesis, Åbo Akademi University, May 2008.

[15] A. Legout, G. Urvoy-Keller and P. Michiardi, "Understanding BitTorrent: An Experimental Perspective", Technical Report (inria-00000156, version 2 - 19 July 2005), INRIA, Sophia Antipolis, July 2005.

[16] J.J.D. Mol, J.A. Pouwelse, M. Meulpolder, D.H.J. Epema and H.J. Sips, "Give-to-Get: Free-riding-resilient Video-on-Demand in P2P Systems", Proc. of SPIE, Multimedia Computing and Networking Conference (MMCN), vol. 6818, article 681804, 2008.

[17] A. Bakker et al, "Tribler Protocol Specification v0.0.2", January 2009, Available from http://www.tribler.org (Accessed August 2010)

[18] OneSwarm Changelog, http://wiki.oneswarm.org/index.php/Changelog (Accessed August 2010)

[19] J. Lv, X. Cheng, Q. Jiang, J. Ye, T. Zhang, S. Lin and L. Wang, "LiveBT: Providing Video-on-demand Streaming Service over BitTorrent Systems", pdcat, pp.501-508, Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2007), 2007

[20] C. Wu, C. Li and J. Ho, "Improving the Download Time of BitTorrent-like Systems", IEEE International Conference on Communications 2007 (ICC 2007), Glasgow, Scotland, June 2007

[21] Spotify, http://www.spotify.com/fi/help/faq/tech/ (Accessed August 2010)

[22] Voddler, http://www.voddler.com/help/topic/2721821230584819787 (Accessed August 2010)

Paper II

Formal Analysis and Verification of Peer-to-Peer Node Behaviour

Petter Sandvik and Kaisa Sere

Originally published in Antonio Liotta, Nikos Antonopoulos, Giuseppe Di Fatta, Takahiro Hara and Quang Hieu Vu (Eds.), *The Third International Conference on Advances in P2P Systems (AP2PS 2011)*, pp. 47–52. IARIA, November 2011.

Formal Analysis and Verification of Peer-to-Peer Node Behaviour

Petter Sandvik[1,2] and Kaisa Sere[1]
[1]Department of Information Technologies, Åbo Akademi University
[2]Turku Centre for Computer Science (TUCS)
Joukahaisenkatu 3–5, 20520 Turku, Finland
{petter.sandvik,kaisa.sere}@abo.fi

***Abstract*—As services and applications move away from the one-to-many relationship of the client-server model towards many-to-many relations such as distributed cloud-based services and peer-to-peer networks, there is a need for a reusable model of how a node could work in such a network. We have constructed a reusable formally derived and verified model of a node in a peer-to-peer network for on-demand media streaming, validated and animated it, and then compared the results with simulations. We have thereby created an approach for analysing peer-to-peer node behaviour.**

Keywords-formal modelling; peer-to-peer; BitTorrent; on-demand streaming.

I. Introduction

There has been a trend in computer software towards a "utility computing vision" [1] in which computer services are accessed without needing to know the specific underlying structure. Rather than the traditional client-server architecture of network services, this vision is largely dependent on many-to-many relations such as distributed cloud-based services and peer-to-peer systems. However, the recent increase in peer-to-peer usage has highlighted a few issues when it comes to development of such services. Testing a peer-to-peer system can be difficult and cumbersome, due to the often large scale and heterogeneous nature of the system. In some cases simulations can be used, but designing a thorough simulation is not an easy task. Furthermore, both testing and simulating these types of systems may require us to emulate the whole network of interacting nodes even if our interest would lie with only one of them, such as when developing a new application designed to interact with a network of existing ones.

Our background in formal methods made us wonder if this problem could be approached from the opposite direction. By this we mean that instead of testing and simulating the whole network to confirm the correct behaviour of one node, we create a formally verified model of one node and then use that model for analysing peer-to-peer node behaviour in general. We will here look at a peer-to-peer on-demand media streaming system, in which content is divided into pieces, distributed between peers using piece selection methods based on BitTorrent on-demand media streaming [2], and then played back in-order. We describe the creation of models for three specific piece selection methods, based on a common reusable formally derived and verified model in Event-B [3]. We then show how ProB [4] can be used to animate our Event-B model, giving results that we can compare with results from simulations. Hence, we show how these different techniques and tools complement each other in the design task.

The rest of this article is organised as follows: In Section II, we describe the Event-B formalism and the tools we have used, and in Section III, we give an overview of on-demand streaming. Section IV details the creation of our formal model. In Section V, we show the results of animation, and compare them with results from previous simulations. We conclude this article in Section VI with discussion and future work.

II. Event-B, the Rodin Platform and ProB

Event-B [5] is a formalism based on Action Systems [6], [7] and the B Method [8]. The primary concept in formal development with Event-B is *models* [5]. A model in Event-B consists of *contexts*, which describe the static parts such as constants and sets, and *machines*, which contain the dynamic parts such as variables, invariants (boolean predicates on the variables), and *events*. An event contains *actions*, which describe how the values of variables change in the event, and *guards*, which are boolean predicates that all must evaluate to `true` before the event can be enabled, i.e., able to execute.

In Event-B development starts from an abstract specification, and the model is then *refined* stepwise into a concrete implementation. In order to achieve a reliable system we use superposition refinement [9], [10] to add functionality while preserving the overall consistency, which means that we add new variables and functionality in such a way that it prevents the old functionality from being disturbed [11]. In order to prove the correctness of each step of the development, we rely on the Rodin Platform [12] tool, which automatically generates *proof obligations*. These proof obligations, which are mathematical formulas that need to be proven in order to ensure correctness, can then be proven either automatically or interactively with the Rodin Platform tool. The choice of Event-B as the formalism to use for a model of this kind was largely due to this integrated tool support.

While the Rodin Platform tool is good for modelling and proving, we would also like to animate, or "execute" our models. This is because the mathematical correctness does not prove that our model does what we wanted it to do [5], and we would also like results that can be compared to those from simulations. ProB [4] is a free-to-use animator

 ISBN: 978-1-61208-173-1

and model checking tool, and supports models from both the B Method and Event-B. While ProB is available as a plug-in for the Rodin Platform tool, we have used the standalone, fully featured version for animation.

III. On-Demand Streaming

Streaming can be described as the transport of data in a continuous flow, in which the data can be used before it has been received in its entirety. There are two different approaches to streaming content; live streaming and on-demand streaming. From an end user perspective live streaming is similar to a broadcast; that is, everyone who receives the media is intended to receive the same content at the same time. On-demand streaming is different, in that it is "essentially playback, as a stream, of pre-recorded content" [13]. This makes on-demand streaming more similar to traditional file transfer. However, on-demand streaming is still "play-while-downloading" and not "open-after-downloading" [14], and traditional file sharing protocols can therefore not be used without modifications. This holds true especially if we look at peer-to-peer file sharing, where content is often transferred out-of-order.

The basis for the peer-to-peer media streaming solution we will look at is the file sharing protocol BitTorrent [15]. In BitTorrent, content is partitioned into pieces of equal size, and by default these pieces are requested out-of-order. By modifying the algorithms used to select the order in which the pieces of the content is requested, i.e. piece selection, BitTorrent can be made to work for streaming content. Several different modifications to the BitTorrent protocol to enable streaming media have been proposed, for instance BiToS [16] and Give-to-Get [17]. We have here chosen to model three different piece selection methods; *sequential*, *rarest-first with buffer* (RFB) and *distance-availability weighted* (DAW). Sequential represents a straightforward streaming solution, requesting pieces in their original order. While this is used for instance when streaming content in the OneSwarm friend-to-friend sharing application [18], BitTorrent contains a tit-for-tat incentive mechanism that requires data to be out-of-order to function as intended, and the sequential method may therefore be of limited use when unknown peers are involved. RFB is a modification of the rarest-first method used in BitTorrent file sharing, where the piece held by the fewest other peers is requested. The addition of a buffer means that a specific number of pieces after the piece currently being played back are requested with the highest priority, thus striving towards always having a certain amount of the content immediately available for playback. and only after that will the rarest piece be requested. DAW [2] tries to strike a balance between requesting rare pieces and pieces that are close to being played back, by calculating priority using the distance (i.e., difference in sequence number between a specific piece and the currently playing one) multiplied with the availability. If there is more than one piece with the same priority, the piece with the lowest piece number, i.e. closest to being played back, will be chosen in both RFB and DAW.

For streaming to work, we note that data cannot be received slower than it should be played back, and this is something that we must take into consideration when creating our model. In the following section, we describe a common Event-B model for the piece selection methods, and refine that model into three specific ones corresponding to the three mentioned piece selection methods.

IV. Modelling with Event-B

Entire peer-to-peer systems and other distributed architectures have been formally modelled [19], [20], [21]. We have created a reusable Event-B model for a node in an on-demand content streaming network [3]. The difference between the two approaches is that instead of looking at the whole network of peers, we model just how one peer looks at the system. Our idea when creating a model is to build it in separate layers, separating the functional parts from each other so that the model could easily be adopted for use with different functionality. Here we focus on modelling the piece selection methods.

As we model our peer-to-peer client as a client for streaming media, we see that three major functions are needed; piece selection (possibly out-of-order), piece transfer (possibly out-of-order) and playback (always in-order). These three functions are independent of each other, but must be performed in this sequence. Hence, pieces must be selected before they are transferred, and pieces must be transferred before they can be played back. An example situation is shown in Fig. 1. As mentioned previously, the content must be transferred at least as fast as it should be played back in order to ensure that streaming works. Therefore, we require that selection will always take place at the same rate as playback or faster; that is, for each time we advance playback we will have selected at least one additional piece.

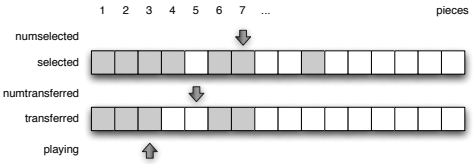


Fig. 1. *The relation between* `selected`, `transferred` *and* `playing`. *The arrows indicate the number of pieces (7 selected, 5 transferred and 3 playing), while the grey squares indicate the specific pieces.*

A. Common Model

We will start with a common model for all three piece selection methods [3], and here we will briefly describe the features of this model. We have two constants, `pieces` and `simreq`, which define how many pieces the content is divided into and the number of simultaneous requests, i.e., the maximum amount of pieces that can be selected but not yet transferred. We have variables for how many pieces we have selected (`numselected`) and exactly which pieces have been selected (`selected`), and similar variables for pieces that have been requested, i.e., transfer started, and for which the transfer has completed. We also keep track of which piece we are playing back and the priority for all pieces, as well as which piece we last updated priority for (`priupd`). The type restrictions of these variables are defined by invariants [3].

 ISBN: 978-1-61208-173-1

machine PieceSelect_M
variables playing completed numselected selected numtransferred transferred numrequested requested priority priupd
invariants
. . .
events
. . .
end

We initialise our common model with having zero pieces, and therefore not having selected, requested or transferred anything. Priorities are set initially for all pieces, but before the priorities are actually used they will be updated. This is done by an abstract event called CHANGE_PRIORITIES. which will be refined later. Initially, this event is enabled as long as we have not updated priorities for all possible pieces to request (@*grd4_1*) and for any p which is a non-zero natural number (@*grd4_2*). The priority of the piece after the previously updated one is then set to p (@*act4_1*), while the value of priupd is set to that piece (@*act4_2*). This means that we will update priorities of all pieces from the currently playing one to the last one.

event CHANGE_PRIORITIES ≙
any p
where
@*grd4_1* priupd < pieces
@*grd4_2* p ∈ ℕ1
then
@*act4_1* priority(priupd+1) ≔ p
@*act4_2* priupd ≔ priupd + 1
end

As the main focus in this model is piece selection, we will now look at the piece selection events. Because we require that content must be transferred faster than it should be played back, we also require that piece selection happens faster than playback. We have modelled this by separating the main flow of the program into two events: SELECT and SELECT_AND_ADVANCE. This means that the action taken in each step can be that of selecting a piece, or selecting a piece and advancing playback. In other words, every time something happens in our model we will select a piece, and some of those times we also advance playback. Naturally, after initialisation we always start with selecting a piece and only after that piece has been transferred could it be possible to advance playback.

The SELECT event is enabled when we have not yet selected as many pieces as we can (@*grd0_1* and @*grd2_4*) and when we have updated priorities for all pieces (@*grd4_5*). The parameter n must also be such that it can represent a piece we have not yet selected (@*grd1_2* and @*grd1_3*) and the priority for piece number n must be less than or equal to the priorities of all other possible pieces (@*grd4_6*). In practice, this means that the maximum priority that can be given to any piece is a numerical value of one, with higher numerical values being less prioritised. It also means that if there is more than one piece with the same priority, and that priority has the lowest numerical value of all priorities given to valid pieces, which one of these pieces to select is not determined. In our case, we will in the next refinements specify which of these pieces to select. However, the way the piece selection is modelled in this common model is actually consistent with the original BitTorrent specification, which does not specify an order when two or more pieces have the same availability [22].

What the SELECT event actually does is to increase the number of selected pieces (@*act0_1*), indicate that the specific piece has been selected (@*act1_2*) and reset the priupd variable so that we can update priorities before the next piece is selected (@*act4_3*).

event SELECT ≙
any n
where
@*grd0_1* numselected < pieces
@*grd1_2* n ∈ playing+1..pieces
@*grd1_3* selected(n) = FALSE
@*grd2_4* numselected − numtransferred < simreq
@*grd4_5* priupd = pieces
@*grd4_6* ∀k · (k ∈ playing+1..pieces ∧ k ≠ n ∧ selected(k) = FALSE ⇒ priority(n) ≤ priority(k))
then
@*act0_1* numselected ≔ numselected + 1
@*act1_2* selected(n) ≔ TRUE
@*act4_3* priupd ≔ playing
end

The SELECT_AND_ADVANCE event is very similar to the SELECT event, with the addition of guards and action concerning advancing playback. Thus, for this event to be enabled we require that we have not played all selected and transferred pieces (@*grd0_a* and @*grd2_c*), and that we have already selected and transferred the piece following the one currently being played back (@*grd1_b* and @*grd3_d*). The actions of this event are identical to the SELECT event, except for the addition of an action increasing the number of the currently playing piece (@*act0_a*) and therefore also requiring the increased value in the action that resets priority updates (@*act4_3*).

event SELECT_AND_ADVANCE ≙
any n
where
@*grd0_1* numselected < pieces
@*grd0_a* playing < numselected
@*grd1_2* n ∈ playing+1..pieces
@*grd1_3* selected(n) = FALSE
@*grd1_b* selected(playing+1) = TRUE
@*grd2_4* numselected − numtransferred < simreq
@*grd2_c* playing < numtransferred
@*grd3_d* transferred(playing+1) = TRUE
@*grd4_5* priupd = pieces
@*grd4_6* ∀k · (k ∈ playing+1..pieces ∧ k ≠ n ∧ selected(k) = FALSE ⇒ priority(n) ≤ priority(k))
then
@*act0_1* numselected ≔ numselected + 1
@*act0_a* playing ≔ playing + 1
@*act1_2* selected(n) ≔ TRUE
@*act4_3* priupd ≔ playing + 1
end

 ISBN: 978-1-61208-173-1

Our model also contains events which are not interesting in this context and therefore not shown here. We have events for pieces being requested and transferred, and in both we require that it is possible to perform the actions enabled by the event, which increase the number of requested or transferred pieces and mark the specific piece number as requested or transferred, respectively. The transfer event thereby updates variables that can be seen in the guards of the SELECT_AND_ADVANCE event. There is also an event that only advances playback after all pieces have been selected. We also have a final event which represents the conditions that must be true for the execution to terminate, which is that all pieces must have been selected, requested, transferred and played back. Only then will we set our variable completed to TRUE.

B. Three Piece Selection Models

Now that we have described our common model, we will take a look at our refined models which represent the use of three different piece selection methods.

1) The Sequential Piece Selection Method: The sequential piece selection method is very simple. Essentially, pieces are selected in order by setting the priority for a piece to its piece number. To model such a piece selection method we can use our common model as a basis, without needing any new variables, constants or events. In fact, the only change is refining the CHANGE_PRIORITIES event. The parameter p from the abstract event is here replaced by its concrete representation, priupd+1, which is the piece number of the piece for which we are changing priority. This necessitates the addition of a witness (*@p*), and we also remove the type guard for p.

```
event CHANGE_PRIORITIES_SEQUENTIAL ≙
    refines CHANGE_PRIORITIES
    when
        @grd4_1 priupd < pieces
    with
        @p priupd+1 = p
    then
        @act4_1 priority(priupd+1) ≔ priupd+1
        @act4_2 priupd ≔ priupd + 1
end
```

2) The Rarest-First Method with Buffer: To model the rarest-first method with buffer (RFB) based on our common abstract model, we need to refine the abstract priority into a concrete one. As described in Section III, the priority in RFB is highest in the buffer, which consists of a fixed number of pieces after the playing one. Outside the buffer, the priority of each piece is set to the availability of that piece. Thus, we add a constant buffersize to describe the size of the buffer, and constants minavail and maxavail to describe the minimum and maximum values for piece availability. Availabilities must be larger than zero, because allowing zero availability for a piece would introduce additional complexity in piece selection and uncertainty as to whether all pieces could actually be transferred. We also need a new variable, availability, to describe the availability of each piece. We also add the following invariant, which states that when we have updated priorities for some but not all pieces, the pieces that we have updated priorities for and that are outside the buffer will have their priorities equal to their availability.

```
@inv5_23 ∀t · (t ∈ playing+1..priupd ∧
      priupd < pieces ∧ t > playing+buffersize
      ⇒ (priority(t) = availability(t)))
```

Initially we set availability to minavail for all pieces. Because the availability is not controlled by us, we need an abstract event which changes the availability of a piece. This event should be enabled independently of piece selection, but not when updating priorities because they depend on the availability. The new event CHANGE_AVAILABILITY is enabled for any valid piece (*@grd5_1*) and availability (*@grd5_2*) whenever we have updated priorities for all pieces (*@grd5_3*), and sets the availability of that piece (*@act5_1*).

```
event CHANGE_AVAILABILITY ≙
    any n a
    where
        @grd5_1 n ∈ 1..pieces
        @grd5_2 a ∈ minavail..maxavail
        @grd5_3 priupd = pieces
    then
        @act5_1 availability(n) ≔ a
end
```

For changing priorities, we refine our abstract event into two separate events, CHANGE_PRIORITIES_BUFFER for setting priorities for pieces in the buffer, and CHANGE_PRIORITIES_RFB for the other pieces. The guard (*@grd5_3*) separates the two different events, ensuring that only one of them is enabled at a time. The parameter p from the abstract event is changed into a concrete one, necessitating the witness (*@p*) and removal of the guard stating the type of p. As can be seen both in the witnesses and in the actions (*@act4_1*), in these events the replacement for p is 1 and the availability of the piece, respectively.

```
event CHANGE_PRIORITIES_BUFFER ≙
    refines CHANGE_PRIORITIES
    when
        @grd4_1 priupd < pieces
        @grd5_3 priupd < playing + buffersize
    with
        @p 1 = p
    then
        @act4_1 priority(priupd+1) ≔ 1
        @act4_2 priupd ≔ priupd + 1
end

event CHANGE_PRIORITIES_RFB ≙
    refines CHANGE_PRIORITIES
    when
        @grd4_1 priupd < pieces
        @grd5_3 priupd ≥ playing + buffersize
    with
        @p availability(priupd+1) = p
    then
        @act4_1 priority(priupd+1) ≔
              availability(priupd+1)
        @act4_2 priupd ≔ priupd + 1
end
```

 ISBN: 978-1-61208-173-1

The SELECT and SELECT_AND_ADVANCE events both gain one guard. This guard corresponds to the requirement that if two pieces have the same priority, the one with the lowest piece number is selected.

```
@grd5_7 ∀j · (j ∈ playing+1..pieces ∧ j ≠ n
    ∧ selected(j) = FALSE ∧
    (priority(n) = priority(j)) ⇒ (n < j))
```

The remaining events do not need refining.

3) The Distance-Availability Weighted Method: When modelling the distance-availability weighted piece selection method (DAW), we can use our experience with modelling RFB as many parts are similar. In this refinement, the contexts of RFB and DAW are identical, and so are the variables. However, the invariant concerning priority (*@inv5_23*) is different. Here, the priorities we have updated outside the buffer should be set to distance times availability [2].

```
@inv5_23 ∀t · (t ∈ playing+1..priupd ∧
    priupd < pieces ∧ t > playing+buffersize
    ⇒ (priority(t) = (t − (playing+buffersize))
    ∗ availability(t)))
```

The CHANGE_AVAILABILITY event introduced in the refinement for RFB is abstract enough that it can be used as-is for DAW as well, but the big difference lies in the CHANGE_PRIORITIES event. Like in RFB, we refine the abstract event from our common model into two different events; one for pieces in the buffer and one for pieces outside the buffer. The CHANGE_PRIORITIES_BUFFER event for pieces in the buffer is, again, identical to the RFB one, as they both assign the highest priority to buffersize pieces after the playing one. However, the event that changes priorities for pieces outside the buffer is different. The parameter p from the abstract event is here replaced with a witness stating the corresponding concrete priority according to the DAW piece selection method.

```
event CHANGE_PRIORITIES_DAW ≙
  refines CHANGE_PRIORITIES
  when
    @grd4_1 priupd < pieces
    @grd5_3 priupd ≥ playing + buffersize
  with
    @p ((priupd+1) − (playing+buffersize))
        ∗ availability(priupd+1) = p
  then
    @act4_1 priority(priupd+1) ≔
        ((priupd+1) − (playing+buffersize))
        ∗ availability(priupd+1)
    @act4_2 priupd ≔ priupd + 1
end
```

The remaining events are identical to the RFB ones, which in some cases means that they are unchanged from the common model. The requirement that if two or more pieces have identical priority the one with the lowest piece number must be selected is also present in DAW.

C. Proof Obligations

Event-B models have certain properties, and when refining a model these properties need to be preserved in order for the model to remain correct. The Rodin Platform tool generates proof obligations, which are the mathematical formulas that need to be proven in order to ensure this correctness, including preserving the invariants and strengthening of the guards of our Event-B models. The Rodin Platform tool can automatically discharge most of these proof obligations by means of automatic provers, but some may need to be proven interactively, which the Rodin Platform also provides the means for. Table I shows the amount of proof obligations generated for each machine by version 2.0.1 of the Rodin Platform tool, and how many of those that needed user interaction. The Rodin Platform tool was used on a computer with a 2.4 GHz Intel Core 2 Duo processor running Mac OS X 10.5.8.

TABLE I
Proof Obligations of our Event-B Model.

Machine	Total Proofs	Interactive Proofs
PieceSelect_M	71	2
PieceSelect_M_SEQ	72	2
PieceSelect_M_RFB	92	3
PieceSelect_M_DAW	93	4

V. Validation, Animation and Comparison

As mentioned in Section II, we have used the standalone, fully featured version of ProB [4]. Due to the way our models are created and ProB interacts with them, memory and processing time constraints have forced us to use smaller values than we would in a real-life situation. Combined with limitations from simulations, this means that we look at the case when the content is divided into 20 pieces, only one request can be outstanding at any time, and the buffer size for RFB and DAW is set to 3. Although smaller than what would be used in a real-world situation, we believe that this is large enough to be noticeable but small enough not to impact the results.

We have compared the results from animating our models in ProB with the results from simple mathematical simulations [2], [13]. These simulations show the behaviour of the piece selection methods for the whole network, and here we chose a network of ten peers starting from scratch and one seed holding all the content. The results show how many of the peers hold each piece after twelve pieces have been selected, when selection happens twice as fast as playback.

Because our Event-B model looks at the network from the point of one node only, we have completed 40 random runs of the animation in ProB, and present the average results from these runs. The minimum availability was set to one and the maximum availability was set to five, and as in the simulation we have stopped to look at the situation after twelve pieces have been selected. Fig. 2 shows the results from both simulation and ProB animation for RFB and DAW piece selection methods.

The results for the sequential piece selection method are not shown, because the results from the simulation are identical

 ISBN: 978-1-61208-173-1

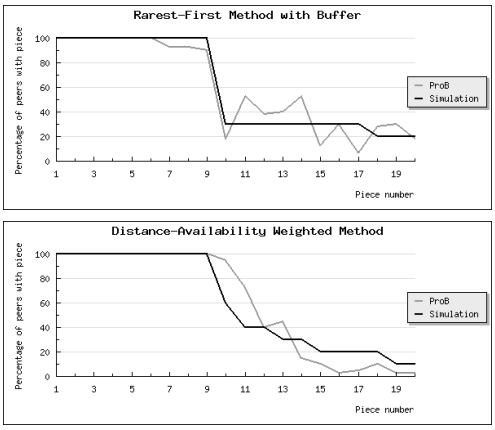


Fig. 2. *The availability of each piece after twelve pieces have been selected, when using RFB and DAW.*

to the ProB results, as should be expected. In both cases, the twelve first pieces are always selected after twelve pieces in total have been selected. For RFB and DAW, we note that the results from simulation and ProB animation are very similar. Some of the differences that exist are due to the fact that the simulations use the actual availability for each piece when calculating priority, while the animation uses random values corresponding to the nondeterminism in our Event-B model. Another difference regards the playback position. In the simulations, playback has always reached exactly piece number six after twelve pieces have been selected, because playback is advanced exactly every other time selection is done. In the animation, we start with selecting a piece, and after that we either select a piece or select a piece and advance playback with equal probability, which leads to variation in playback position but a mathematical average of 5.5. This means that unlike in the simulation, in the ProB animation an RFB node may not always have selected the 9 first pieces, and this is visible in Fig. 2. The very nature of DAW makes it unlikely, although not impossible, for the same thing to happen with DAW.

VI. Conclusions and Future Work

We have created a formally constructed and verified Event-B model of a node in a peer-to-peer content streaming network. This model we have then refined and animated, and the results have been compared with simulations. Using the same piece selection methods, our formal model of one peer has given us similar average results as a simulation of the whole network of peers. While we ran the ProB animation using smaller figures than would be used in a real-world situation, we still believe that our results show the added value that our method creates. By itself or together with simulations, animation of a formally derived and verified model can be used as an approach to analysing and verifying peer-to-peer node behaviour.

As our model is reusable, future work could include refining our model for other piece selection methods and extending the models and comparisons to other peer-to-peer systems besides an on-demand streaming one. Another possible direction would be refining our formal models to include the network structure in order to facilitate analysing aspects that require knowledge of more than just one node.

References

[1] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation Computer Systems*, vol. 25, pp. 599–616, 2009.

[2] P. Sandvik and M. Neovius, "The Distance-Availability Weighted Piece Selection Method for BitTorrent: A BitTorrent Piece Selection Method for On-Demand Streaming," in *Proceedings of AP2PS '09*, October 2009.

[3] P. Sandvik, K. Sere, and M. Waldén, "An Event-B Model for On-Demand Streaming," Turku Centre for Computer Science (TUCS), Tech. Rep. 994, December 2010, http://tucs.fi/publications/insight.php?id=tSaSeWa10a (Accessed September 2011).

[4] "The ProB Animator and Model Checker," http://www.stups.uni-duesseldorf.de/ProB/ (Accessed September 2011).

[5] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[6] R.-J. Back and K. Sere, "From Modular Systems to Action Systems," *Software - Concepts and Tools*, vol. 13, pp. 26–39, 1996.

[7] M. Waldén and K. Sere, "Reasoning About Action Systems Using the B-Method," *Formal Methods in Systems Design*, vol. 13, pp. 5–35, 1998.

[8] J.-R. Abrial, *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[9] R.-J. Back and R. Kurki-Suonio, "Decentralization of Process Nets with Centralized Control," in *Proceedings of the 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 1983, pp. 131–142.

[10] S. Katz, "A Superimposition Control Construct for Distributed Systems," *ACM Transactions on Programming Languages and Systems*, vol. 15(2), pp. 337–356, April 1993.

[11] K. Sere, "A Formalization of Superposition Refinement," in *Proceedings of the 2nd Israel Symposium on the Theory and Computing Systems*, June 1993.

[12] "Event-B and the Rodin Platform," http://www.event-b.org/ (Accessed September 2011).

[13] P. Sandvik, "Adapting Peer-to-Peer File Sharing Technology for On-Demand Media Streaming," Master's thesis, Åbo Akademi University, May 2008.

[14] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On Peer-to-Peer Media Streaming," in *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, 2002.

[15] B. Cohen, "BitTorrent - A New P2P App," Yahoo eGroups, http://finance.groups.yahoo.com/group/decentralization/message/3160 (Accessed September 2011).

[16] A. Vlavianos, M. Iliofotou, and M. Faloutsos, "BiToS: Enhancing BitTorrent for Supporting Streaming Applications," in *9th IEEE Global Internet Symposium 2006*, April 2006.

[17] J. Mol, J. Pouwelse, M. Meulpolder, D. Epema, and H. Sips, "Give-to-Get: Free-riding-resilient Video-on-Demand in P2P Systems," in *Multimedia Computing and Networking 2008, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 6818*, 2008.

[18] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson, "Privacy-Preserving P2P Data Sharing with OneSwarm," in *SIGCOMM'10*, August–September 2010, pp. 111–122.

[19] L. Yan and J. Ni, "Building a Formal Framework for Mobile Ad Hoc Computing," in *Proceedings of the International Conference on Computational Science (ICCS'04)*, June 2004.

[20] L. Yan, "A Formal Architectural Model for Peer-to-Peer Systems," in *Handbook of Peer-to-Peer Networking 2010 Part 12*, X. Shen, H. Yu, J. Buford, and M. Akon, Eds. Springer US, 2010, pp. 1295–1314.

[21] M. Kamali, L. Laibinis, L. Petre, and K. Sere, "Self-Recovering Sensor-Actor Networks," in *FOCLASA*, 2010.

[22] B. Cohen, "Incentives Build Robustness in BitTorrent," in *1st Workshop on Economics of Peer-to-Peer Systems*, June 2003.

 ISBN: 978-1-61208-173-1

Paper III

Node Coordination in Peer-to-Peer Networks

Luigia Petre, Petter Sandvik, and Kaisa Sere

Originally published in Marjan Sirjani (Ed.), *COORDINATION 2012*, Lecture Notes in Computer Science Vol. 7274, pp. 196–211. Springer-Verlag GmbH Berlin Heidelberg, June 2012.

Node Coordination in Peer-to-Peer Networks

Luigia Petre[1], Petter Sandvik[1,2], and Kaisa Sere[1]

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

Abstract. Peer-to-peer networks and other many-to-many relations have become popular especially for content transfer. To better understand and trust these types of networks, we need formally derived and verified models for them. Due to the large scale and heterogeneity of these networks, it may be difficult and cumbersome to create and analyse complete models. In this paper, we employ the modularisation approach of the Event-B formalism to model the separation of the functionality of each peer in a peer-to-peer network from the network structure itself, thereby working towards a distributed, formally derived and verified model of a peer-to-peer network. As coordination aspects are fundamental in the network structure, we focus our formalisation effort in this paper especially on these. The resulted approach demonstrates considerable expressivity in modelling coordination aspects in peer-to-peer networks.

1 Introduction

In recent years, there has been a trend of moving away from the traditional client-server model in network software towards peer-to-peer networks and other many-to-many relations. Especially when it comes to large scale content transfer, peer-to-peer applications and protocols such as BitTorrent [8] have become popular [24], and even found their way into electronic appliances such as network routers [6] and television sets [26]. In short, the paradigm switch from client-server communication models to BitTorrent-supporting networks amounts to enabling "clients" that are already downloading e.g., video streams, to also become "servers" for other potential clients that may download the same content. The participation of every peer in content communication provides a tremendous increase in the communication efficiency, in the communication model flexibility, and in the content availability. It is therefore highly beneficial to have a thorough understanding of this communication paradigm, to uncover its potential weaknesses and recognise how to avoid them.

Peer-to-peer networking proposes a mixed coordination model among peers. At first sight, it resembles data-based coordination, such as distributed tuple spaces, in that one peer enumerates in a webpage its downloadable material and another peer starts downloading the material of interest, found via the webpage or via a special server called tracker. However, even during downloading,

M. Sirjani (Ed.): COORDINATION 2012, LNCS 7274, pp. 196–211, 2012.

the second peer also becomes a data provider of that material, solely due to its downloading and without any enumeration of downloadable material in a webpage. This resembles event-based coordination where communication between processes (peers) is enabled by events generated when certain state changes appear. This is also reminiscent of the publisher-subscriber model of coordination. This partial adherence of peer-to-peer networking to several coordination models is currently not singular. In [18], the authors propose the separation of the coarse grained (coordination) control flow into several event handlers that coordinate (via events) the mobile applications. In their turns, the event handlers need to communicate with each other, typically implicitly, via shared data. Coordination and concurrency are studied in the context of Prolog [25] by decoupling logic engines and multithreads for efficiency; cooperative constructs are then illustrated for both Linda [7] blackboards and publish/subscribe models. Real-time coordination in dataflow networks is typically asynchronous, but in [16], coordination patterns are proposed which combine synchrony and asynchrony. All these models simply try to address the ever-increasing complexity of contemporary software-intensive systems from various viewpoints. However, due to combining several aspects of several coordination models, peer-to-peer networking is a rather complicated model to analyse. In this paper we focus on this analysis problem.

In order to gain a thorough understanding of peer-to-peer networking, we develop and analyse models of a peer-to-peer media distribution system. In particular, in this paper we focus on modelling how peers in a such a system could discover and interact with each other, i.e., we model inter-peer relations as the basis of the peer-to-peer coordination model. In swarm-like peer-to-peer systems, where peers interact only when interested in the same content, a peer that is unable to receive incoming connections, for instance when it is behind a firewall, is at a serious disadvantage compared to other peers [10]. Extensions to the original BitTorrent protocol such as peer exchange (PEX) and distributed hash tables (DHT) [17] have been developed to alleviate this problem, and we need a reusable, extendable model of peer discovery and connectivity to be able to model these. Peer-to-peer systems and other distributed architectures have been formally modelled before [15,28,29], but our focus here is on creating a reusable formal model of inter-peer relations using BitTorrent as our model protocol.

Based on the formal modelling of peer-to-peer relations, we make the following contributions:

- We propose a formal model for analysing properties of peer-to-peer relations and networking.
- We distribute this model and the proven properties as a correct development from the initial model.
- We put forward the dual coordination nature of the distributed model (both data-driven and control-oriented) and the further applicability of our employed formal methodology.

We develop our models based on the Event-B formal method [2], which offers excellent tool support in form of the Rodin platform [3,11]. When developing

models in Event-B, the primary concept is that of abstraction [2], as *models* are created from abstract specifications and then refined stepwise towards concrete implementations. We prove the correctness of each step of the development using the Rodin platform, which automatically generates *proof obligations*. These are mathematical formulas to prove in order to ensure correctness; the proving can be done automatically or interactively using the Rodin platform tool. The immediate feedback from the provers makes it possible to adapt our model to better suit automatic proving, and this ability to interleave modelling and proving is a big advantage of development in Event-B using the Rodin platform. Event-B is currently extending to also incorporate *modularisation methodology* [13]. This essentially amounts to proposing distributed versions for various models and proving the correctness of the distribution via refinement. Consider the example of a peer connection operation involving two nodes. We can specify this feature in Event-B typically within one module (called *machine*) that has data and operations on the data; we can also model various properties of the module and prove their correctness. However, at the implementation phase, the peer connection operation typically involves two modules, corresponding to the two connecting peers that synchronise with each other, so that each peer adds the required reference to the other. The modularisation methodology allows the transformation of the modelled peer connection operation into a distributed addition of links among the two peers.

We proceed as follows. In Section 2 we describe the Event-B formalism and its modularisation extension. In Section 3 we introduce our inter-peer relation modelling and in Section 4 we present our modular approach to this. We elaborate on our contribution in Section 5. We conclude this paper in Section 6 with discussion of our findings as well as future work.

2 Event-B and Its Modularisation Approach

In this section we overview Event-B and its modularisation approach to the extent needed in this paper.

2.1 Event-B

Event-B [2] is a state-based formal method focused on the stepwise development of correct systems. This formalism is based on Action Systems [5,27] and the B-Method [1]. In Event-B, the development of a model is carried out step by step from an abstract specification to more concrete specifications. The general form of an Event-B model is illustrated in Fig. 1. Models in Event-B consist of *contexts* and *machines*. A context describes the static part of a model, containing sets and constants, together with axioms about these. A machine describes the dynamic part of a model, containing variables, invariants (boolean predicates on the variables), and events, that evaluate (via event *guards*) and modify (via event *actions*) the variables. The guard of an event is an associated boolean predicate on the variables, that determines if the event can execute or not. The action of

an event is a parallel composition of either deterministic or non-deterministic assignments. Computation proceeds by a repeated, non-deterministic choice and execution of an enabled event (an event whose guard holds). If none of the events is enabled then the system deadlocks. The relationship *Sees* between a machine and its accompanying context denotes a structuring technique that allows the machine access to the contents of the context.

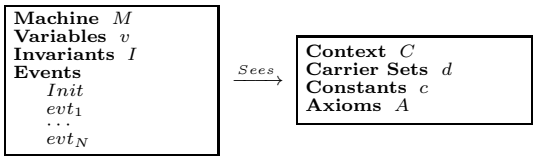


Fig. 1. A machine M and a context C in Event-B

The semantics of Event-B actions is defined using *before-after (BA) predicates* [2,3]. A before-after predicate describes a relationship between the system states before and after the 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. The full list of proof obligations can be found in [2].

System Development. Event-B employs a top-down refinement-based approach to the formal system development. The development starts from an abstract system specification that models some 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. This type of refinement is called *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 invariants are called *gluing invariants*.

In order to prove the correctness of each step of the development, a set of proof obligations needs to be discharged. Thus, in each development step we have mathematical proof that our model is correct. The model verification effort and, in particular, the automatic generation and proving of the required proof obligations, are significantly facilitated by the provided tool support – the Rodin platform [3,4].

2.2 The Event-B Modularisation Approach

Recently the Event-B language and tool support have been extended with a possibility to define *modules* [13,12,21] – i.e., components containing groups of callable atomic operations. Modules can have their own external (i.e., global) and internal (i.e., local) state and invariant properties. An important characteristic

of modules is that they can be developed separately and, when needed, composed with the main system.

A module description consists of two parts – a *module interface* and a *module body*, the latter being an Event-B machine. 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 O_i, characterised by their pre- and post-conditions, as shown in Fig. 2.

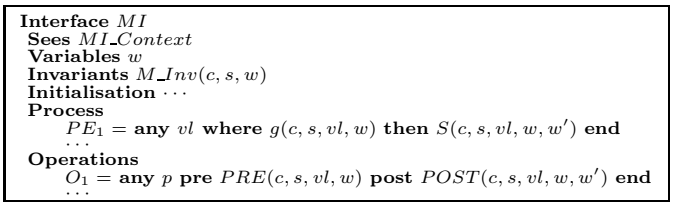


Fig. 2. Interface Component

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

A formal module development starts with the design of an interface. Once an interface is defined, it is not further developed. 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 that 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 (which is 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; we distinguish among these instances using a certain $prefix$. Different instances of a module operate on disjoint state spaces. Via different instantiation 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 allow 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.

3 Modelling Inter-peer Relations

We illustrate the first three steps of our development of a formal model for inter-peer relations in Fig. 3. In this section we shortly describe this Event-B model in order to facilitate an easier understanding of the modularised model described in the next section. More details can be found in our technical report [20].

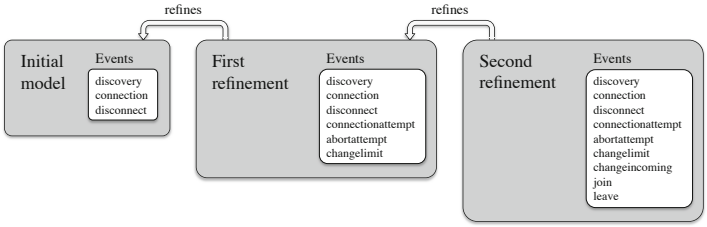


Fig. 3. Model Development

Our initial model is very abstract, with only two major functions. The first concerns one peer becoming aware of other peers. In a peer-to-peer network such as BitTorrent, this would correspond to receiving a list of other peers from a tracker, i.e., a server that keeps track of which peers are involved in sharing a particular content. However, at this stage we are not interested in the specifics of how this subset of all peers is retrieved, only that there is a way of peers to discover other peers. We also note that the tracker is an instantiation of the publish/subscribe coordination model. The second major function is to create a connection between a peer and another peer, where the first peer must be aware of the second but not necessarily vice versa. To model these functions, we define relations between peers, assuming peers are represented by natural numbers for simplicity. An "awareness" relation from 1 to 2 thereby means that peer 1 is aware of peer 2, which is different from a relation from 2 to 1. For the "connection" relation, we note that in practice we only have one connection between two peers, because in peer-to-peer networks such as those based on BitTorrent, connections are symmetrical and traffic can flow in both directions [9]. For that reason, we allow only one connection per peer pair here, e.g., if a "connection" relation exists from 1 to 2 we do not allow one from 2 to 1.

Our initial model is therefore composed of the following events, besides the obligatory *initialisation* event: *discovery*, which creates "awareness" relations from a peer to a subset of other peers, *connection*, which creates a "connection" relation between a peer and another if there is an "awareness" relation from the first to the second, and *disconnect*, which removes an existing "connection" relation between two peers. This disconnection could occur because of network issues or because the peer has decided to no longer participate in the swarm. However, peers also close connections that have had no traffic for a while; Iliofotou et al claim that the differences in download speed between BitTorrent clients can be partly attributed to differences in when they decide to close connections [14]. For this reason it is important for us to model a *disconnect* event that later can be refined into different types of disconnection events. The situation in which peers become unaware of each other does not exist in the actual peer-to-peer networks we are interested in, and therefore there is no need for an event that models such a situation.

For our first refinement step, we limit the amount of connections a peer can have, because otherwise every peer would eventually end up being connected to all the other peers. While this would be possible when the number of peers is low, it would be unrealistic for a large system, and we therefore introduce a connection limit specific to each peer. This means that a connection between two peers may not always be possible, and therefore we also need to modify our connection functionality. Because peers do not know whether another peer can accept their connection or not, we replace our single connection event with two events. The *connectionattempt* event takes a peer whose connection limit has not been reached and another peer that the first peer is aware of but not connected to, and adds a "connection attempt" relation from the first peer to the second one. The *connection* event here takes a peer whose connection limit has not been reached and another peer such that there is a "connection attempt" relation from the second to the first, and creates a "connection" relation from the second to the first while removing the corresponding "connection attempt" relation. We also add another event, *abortattempt*, for aborting a connection attempt, which in practice would happen after a time limit. Because the connection limit is not necessarily constant and can vary between peers, we also add the abstract *changelimit* event describing how the limit may change. The total amount of connections for a peer, specified by the variable *connections*, is here taken to be the sum of the amount of "connection" relations to and from the peer, and the amount of "connection attempt" relations originating from the peer. This means that the limit on connections is a limit on the amount of simultaneous active successful connections and unsuccessful connection attempts.

In the second refinement step we introduce the concept of peers not being able to accept incoming connections, i.e., not being able to have "connection" relations from another peer to itself. First we achieve this in an abstract way, by simply having a boolean variable for each peer and checking the value of that variable before allowing the connection to be created. We add the abstract event *changeincoming* to be able to change the value of this boolean variable for

each peer. Later we can refine this situation by specifying a set of more complex relations, such as in the real-life situation where two peers are behind the same firewall and thereby able to accept incoming connections from each other but not from other peers. Furthermore, we refine our model to include *join* and *leave* events for when peers join and leave the swarm, respectively. To reduce the complexity of our model, we specify that all the connections to and from a peer, as well as all connection attempts made by the peer, must be removed before the peer can leave. This can be seen in the following Event-B code:

EVENT $leave \mathrel{\widehat{=}}$
  any
    $peer$
  where
    grd1 : $peer \in peers \land peer \in onlinepeers$
    grd2 : $\forall p, r \cdot (\{p \mapsto r\} \in connection) \Rightarrow (p \neq peer \land r \neq peer)$
    grd3 : $\forall p, r \cdot (\{p \mapsto r\} \in connectionattempt) \Rightarrow (p \neq peer)$
  then
    act1 : $onlinepeers := onlinepeers \setminus \{peer\}$
  end

So far, we have described a monolithic model of inter-peer relations in a peer-to-peer network. Our next step is to use the modularisation approach described in Section 2.2 to separate the internal functionality of a peer from the coordinating functionality of the network structure.

4 Modularising Inter-peer Relations

Our intent with modularising our model of inter-peer relations is to separate the internal functionality of each peer from the functionality of the network itself; this makes the peers, in a sense, independent of other peers. As we specify the interface that a peer presents to the coordinating network, we can continue to refine and implement the peer separately from the network coordination structure. Therefore, we need to consider which events from our previous model should be implemented in the peer module and which in the Event-B machine specifying the network coordination.

We note that the events *changelimit* and *changeincoming* affect only one peer at a time, and thus should be modelled as processes internal to the peer. Likewise, the *discovery* event only adds to one peer's view, and although it could be argued that this is an event concerning network coordination, nothing specifies that this event needs to invoke the network at all. In BitTorrent, for instance, peer discovery never depends on how peers connect to each other, and therefore it should be seen as a process internal to the peer in this context. The *join* and *leave* events also only affect one peer's status, because we require that the *leave* event is enabled only when the peer has no connections and no connection attempts. This is also reflected in the identically named process in the peer interface, which can be compared to the *leave* event shown in Section 3.

PROCESS $leave \mathrel{\widehat{=}}$
  when
    grd1 : $isonline = TRUE$
    grd2 : $connection = \varnothing \land connectionattempt = \varnothing$

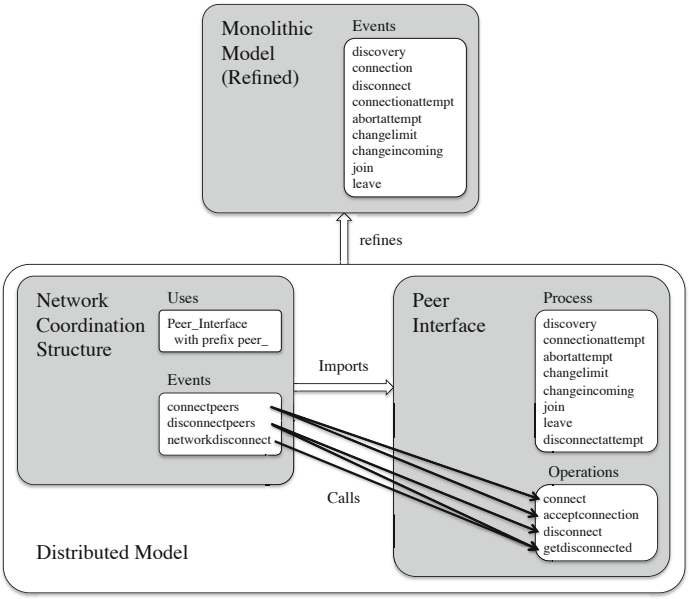


Fig. 4. Decomposition refinement

```
then
    act1 : isonline := FALSE
end
```

Regarding the *connectionattempt* and *abortattempt* events, we note that these events model the intent of one peer, and thus should be modelled as an event internal to the peer, although the variables modified will be read by the network coordination structure. The remaining events *connection* and *disconnect* require coordination between peers, and thus we will describe in more detail how the equivalent functionality is implemented in the distributed model. The overall structure of this decomposition refinement can be seen in Fig. 4.

As mentioned in Section 2.2, we use the **USES** clause to import an interface into an Event-B machine, specifying the name of the interface, the indexing set, and the prefix used to access varibles and operations from the interface.

USES *Peer_Interface* (peers) with prefix *peer_*

We previously added a guard specifying that a *connectionattempt* must be done before a *connection*. Here, the *connectionattempt* is internal to the peer, and the network coordination structure has an event *connectpeers*. Given two different online peers s and t, who are not connected to each other, and where s has made a connection attempt to t and t can accept an incoming connection, the operation *acceptconnection* of peer t is called with the argument s, and likewise the operation *connect* of peer s is called with the argument t.

EVENT *connectpeers* $\widehat{=}$
  any
    s t
  where
    grd1 : $s \in peers \wedge t \in peers \wedge s \neq t$
    grd2 : $t \in peer\_connectionattempt(s) \wedge peer\_acceptincoming(t) = TRUE$
    grd3 : $t \notin peer\_connection(s) \wedge s \notin peer\_connection(t)$
    grd4 : $peer\_isonline(s) = TRUE \wedge peer\_isonline(t) = TRUE$
    grd5 : $peer\_connections(t) < peer\_connectionlimit(t)$
  then
    act1 : $void1 := peer\_acceptconnection(t)(s)$
    act2 : $void2 := peer\_connect(s)(t)$
  end

The variables *void1* and *void2* used here are of the type *VOID*, which is used when an operation call has no return value.

There is a difference between the *acceptconnection* and *connect* operations of the peer interface, in that the former is to be called on a peer that has not made a connection attempt, while the latter is to be called on a peer who has made one. This means that among the preconditions for the *acceptconnection* operation is that the peer must accept incoming connections and must not have reached its connection limit.

OPERATION *acceptconnection* $\widehat{=}$
  any
    $dest$
  pre
    pre1 : $dest \in peers \wedge dest \notin connection \wedge dest \notin connectionattempt$
    pre2 : $connections < connectionlimit$
    pre3 : $acceptincoming = TRUE \wedge isonline = TRUE$
  return
    $void$
  post
    post1 : $connection' = connection \cup \{dest\}$
    post2 : $connections' = connections + 1$
    post3 : $void' :\in VOID$
  end

For the peer receiving the *connect* operation call the amount of connections was already increased when making the connection attempt, and therefore should not be increased here. However, as a peer is added to the set of connections, it must also be removed from the set of connection attempts.

OPERATION *connect* $\widehat{=}$
  any
    $dest$
  pre
    pre1 : $dest \in peers \wedge dest \notin connection \wedge dest \in connectionattempt$
    pre2 : $isonline = TRUE$
  return
    $void$
  post
    post1 : $connection' = connection \cup \{dest\}$
    post2 : $connectionattempt' = connectionattempt \setminus \{dest\}$
    post3 : $void' :\in VOID$
  end

In our monolithic model, the *disconnect* event simply disconnected two peers that were connected. However, we note that in this refinement we need to separate disconnection into two cases; the first of which concerns a peer actively wanting to disconnect from another peer, and another case when the disconnection happens without the intent of any of the peers involved. In the first case,

we will handle it similarly to the connection process. In the peer interface, we specify a new process, *disconnectattempt*, which modifies a variable that will be read by the network coordination machine. When the prerequisites are fulfilled, i.e., when two distinct peers are connected and one of them has made a disconnection attempt concerning the other, the event *disconnectpeers* in the machine then calls the *disconnect* operation on the originating peer and *getdisconnected* on the other.

EVENT $disconnectpeers \mathrel{\widehat{=}}$
  any
    $p\ r$
  where
    grd1 : $p \in peers \wedge r \in peers \wedge p \neq r$
    grd2 : $r \in peer\_connection(p) \wedge p \in peer\_connection(r)$
    grd3 : $r \in peer\_disconnectionattempt(p)$
  then
    act1 : $void1 := peer\_getdisconnected(r)(p)$
    act2 : $void2 := peer\_disconnect(p)(r)$
  end

In the peer interface, the two operations *getdisconnected* and *disconnect* are very similar. In the first, the peer must remove the connection to a specific peer for which no "disconnection attempt" has been created, and decrease the number of total connections.

OPERATION $getdisconnected \mathrel{\widehat{=}}$
  any
    $dest$
  pre
    pre1 : $dest \in peers \wedge dest \in connection \wedge dest \notin disconnectionattempt$
    pre2 : $connections > 0$
  return
    $void$
  post
    post1 : $connection' = connection \setminus \{dest\}$
    post2 : $connections' = connections - 1$
    post3 : $void' :\in VOID$
  end

For the *disconnect* operation to be enabled, there must be a "disconnection attempt", but otherwise the preconditions are the same as in the *getdisconnected* operation. The postconditions are also identical to the previously described operation, with the addition that the "disconnection attempt" must also be removed.

OPERATION $disconnect \mathrel{\widehat{=}}$
  any
    $dest$
  pre
    pre1 : $dest \in peers \wedge dest \in connection \wedge dest \in disconnectionattempt$
    pre2 : $connections > 0$
  return
    $void$
  post
    post1 : $connection' = connection \setminus \{dest\}$
    post2 : $connections' = connections - 1$
    post3 : $void' :\in VOID$
    post4 : $disconnectionattempt' = disconnectionattempt \setminus \{dest\}$
  end

As we mentioned, two peers can get disconnected not only by their own intent but also because of external factors. We model this in the network structure machine with the event *networkdisconnect*. This event simply calls the operation

getdisconnected on each of the two peers, with the other peer as argument, with the prerequisite that the peers must be connected to each other but not have tried to disconnect of their own intent.

EVENT $networkdisconnect \mathrel{\widehat{=}}$
  any
    u v
  where
    grd1 : $u \in peers \land v \in peers \land u \neq v$
    grd2 : $v \in peer\_connection(u) \land u \in peer\_connection(v)$
    grd3 : $v \notin peer\_disconnectattempt(u) \land u \notin peer\_disconnectattempt(v)$
  then
    act1 : $void1 := peer\_getdisconnected(u)(v)$
    act2 : $void2 := peer\_getdisconnected(v)(u)$
  end

The intended goal when creating any formal model such as ours is to be able to prove various properties in the system being modelled. For our monolithic model, all generated proof obligations can be easily discharged using the proving environment of the Rodin platform tool [11]. As the Modularisation plugin includes proof generation and proving support for its extensions to the Event-B language [13], many of the properties that we can prove in the original monolithic model we can also prove in the distributed model. We put forward an example of the transformation of a property to prove from the monolithic to the distributed model in the following section.

Using this modularisation technique to do decomposition refinement increases the complexity of the model, which makes proving more difficult. This applies equally to the automatic proof obligation discharging and interactive proving in the Rodin platform tool. As the tool and the Modularisation plugin evolves, we hope that it will enable us to develop our models further than what is currently possible.

5 Discussion

In this section we summarise the contributions of this paper.

First, we propose a (stepwise developed) monolithic model for inter-peer relations in a peer-to-peer network. This model has a (simple) state consisting of the values of the variables and a set of events that can all access and modify the state. Due to the high level of abstraction, we can formulate and prove various properties about our model. For instance, we have an invariant stating that any peer that is connected to another peer, i.e., has a "connection" relation to it, cannot have a "connection attempt" relation to the same peer, put forward below. Coordination between peers is centralised and endogenous, for instance the event *connectionattempt* coordinates the establishment of a "pre-connection" relation and the event *connection* coordinates the establishment of a real "connection" when the proper conditions for it are met.

$$\forall p, r \cdot (\{p \mapsto r\} \in connection) \Rightarrow (\{p \mapsto r\} \notin connectionattempt) \qquad (1)$$

Second, we refine the monolithic model into a distributed one, in which we separate the coordination between peers from the internal actions (computation) of

the peers. Coordination is now exogenous, modelled by the events *connectpeers*, *disconnectpeers*, and *networkdisconnect*. The properties proven in the monolithic model evolve as well, as exemplified in the following. In the peer interface of the distributed model, we are modelling the interface of one peer, and therefore that peer does not need to be included. Thus, the corresponding invariant in the peer interface states that each peer that we have a connection to must not be in the set of connection attempts.

$$\forall p \cdot p \in peers \wedge p \in connection \Rightarrow p \notin connectionattempt \tag{2}$$

In both cases, it is of course also trivial to prove the inverse implication, i.e., that a connection attempt between two peers implies that there is no existing connection between the two.

We note that the coordination in the distributed model is rather sophisticated. The coordinator (the network coordination structure) only reads the value of the coordinated peer state (via external variables such as *peer_connection(u)* in event *networkdisconnect*). The peer state is only modified via the nodes' own actions, as described in the operation *getdisconnected*. We can also argue for the coordination paradigm displayed by our modelling to be of a mixed nature. On one hand, the external variables of the peers model a distributed (tuple) space; the coordinator only acts based on reading this space, hence a data-driven coordination. On the other hand, the execution of the coordination actions is not performed directly on the data, but via procedure calls mechanisms, hence, a control-oriented coordination model.

6 Conclusions

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. With the aim of modelling and analysing a whole, fully featured peer-to-peer media distribution system, we have used Event-B to model inter-peer relations in a BitTorrent-like peer-to-peer network. We have started from an abstract specification and stepwise introduced functionality so that the proving effort remains reasonable. For instance, we could have introduced the *join* and *leave* events already in the first model; however, this would have generated unnecessary proving at an abstract level.

Our focus has been on creating a model of a peer-to-peer system in a way that allows it to be reused and extended for different protocol additions, while keeping the reliability of the system intact. This gives us a foundation from which we can develop a well behaving and scalable peer-to-peer media distribution system. Our goal is to have all the parts, from the network structure up to the content playback, formally modelled and verified. We have previously modelled different parts of such a system, including algorithms for acquiring pieces of media content [22,23] and parts of a video decoding process [19].

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 [12]. 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 complexity of distributed systems verification and gradually derive an implementation with the desired properties and behaviour [2].

With respect to proving properties about models, our strategy is very useful: we formulate and prove properties for the monolithic model and then we develop the distributed model from the monolithic one so that the properties remain valid. This is however not new, as it has been proposed in a number of earlier works, for instance in [5]. With respect to the coordination paradigm, we consider that modularisation in Event-B provides a very interesting methodology for emphasising the separation of the coordination features from the computation ones. This is especially useful in the context of the Rodin tool platform [11] that can significantly improve the property proving effort and thus puts forward our approach to coordination as a practical one.

As future work, we plan to develop the peer-to-peer networking models into an Event-B theory. This means that we can then model specific peer-to-peer networks simply by instantiating them from the theory, much like declaring data types. Hence, we envision a language construct for modern network architectures. With this, we stress once more the reuse potential of our proposal.

Acknowledgements. We would like to thank Alexei Iliasov for helping us to better understand the Modularisation plugin.

References

1. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press (1996)
2. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
3. 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. International Journal on Software Tools for Technology Transfer (STTT) 12(6), 447–466 (2010)
4. Abrial, J.R., Butler, M., Hallerstede, S., Voisin, L.: An Open Extensible Tool Environment for Event-B. In: Liu, Z., Kleinberg, R.D. (eds.) ICFEM 2006. LNCS, vol. 4260, pp. 588–605. Springer, Heidelberg (2006)
5. Back, R., Kurki-Suonio, R.: Decentralization of Process Nets with Centralized Control. In: Proceedings of the 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, pp. 131–142 (1983)
6. Belkin Play N600 HD Wireless Dual-Band N+ Router F7D8301, `http://www.belkin.com/IWCatProductPage.process?Product_Id=522112` (accessed April 2012)

7. Carriero, N., Gelernter, D.: Data Parallelism and Linda. In: Banerjee, U., Gelernter, D., Nicolau, A., Padua, D.A. (eds.) LCPC 1992. LNCS, vol. 757, pp. 145–159. Springer, Heidelberg (1993)
8. Cohen, B.: Incentives Build Robustness in BitTorrent. In: 1st Workshop on Economics of Peer-to-Peer Systems (June 2003)
9. Cohen, B.: The BitTorrent Protocol Specification (January 2008), `http://www.bittorrent.org/beps/bep_0003.html` (accessed April 2012)
10. D'Acunto, L., Meulpolder, M., Rahman, R., Pouwelse, J., Sips, H.: Modeling and Analyzing the Effects of Firewalls and NATs in P2P Swarming Systems. In: IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum, IPDPSW (2010)
11. Event-B and the Rodin Platform, `http://www.event-b.org/` (accessed April 2012)
12. Iliasov, A., Laibinis, L., Troubitsyna, E., Romanovsky, A.: Formal Derivation of a Distributed Program in Event B. In: Qin, S., Qiu, Z. (eds.) ICFEM 2011. LNCS, vol. 6991, pp. 420–436. Springer, Heidelberg (2011)
13. Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A., Varpaaniemi, K., Ilic, D., Latvala, T.: Supporting Reuse in Event B Development: Modularisation Approach. In: Frappier, M., Glässer, U., Khurshid, S., Laleau, R., Reeves, S. (eds.) ABZ 2010. LNCS, vol. 5977, pp. 174–188. Springer, Heidelberg (2010)
14. Iliofotou, M., Siganos, G., Yang, X., Rodriguez, P.: Comparing BitTorrent Clients in the Wild: The Case of Download Speed. In: Freedman, M.J., Krishnamurthy, A. (eds.) Proceedings of the 9th International Workshop on Peer-to-Peer Systems, IPTPS 2010. USENIX (April 2010)
15. Kamali, M., Laibinis, L., Petre, L., Sere, K.: Self-Recovering Sensor-Actor Networks. In: Mousavi, M., Salan, G. (eds.) Proceedings of the Ninth International Workshop on the Foundations of Coordination Languages and Software Architectures, FOCLASA 2010, vol. 30, pp. 47–61. EPTCS (2010)
16. Kemper, S.: Compositional Construction of Real-Time Dataflow Networks. In: Clarke, D., Agha, G. (eds.) COORDINATION 2010. LNCS, vol. 6116, pp. 92–106. Springer, Heidelberg (2010)
17. Loewenstern, A.: DHT Protocol (2008), `http://www.bittorrent.org/beps/bep_0005.html` (accessed April 2012)
18. Lombide Carreton, A., D'Hondt, T.: A Hybrid Visual Dataflow Language for Coordination in Mobile Ad Hoc Networks. In: Clarke, D., Agha, G. (eds.) COORDINATION 2010. LNCS, vol. 6116, pp. 76–91. Springer, Heidelberg (2010)
19. Lumme, K., Petre, L., Sandvik, P., Sere, K.: Towards Dependable H.264 Decoding. In: Ahmed, N., Quercia, D., Jensen, C.D. (eds.) Workshop Proceedings of the Fifth IFIP WG 11.11 International Conference on Trust Management (IFIPTM 2011), pp. 325–337. Technical University of Denmark (June 2011)
20. Petre, L., Sandvik, P., Sere, K.: A Modular Approach to Formal Modelling of Peer-to-Peer Networks. Tech. Rep. 1039, Turku Centre for Computer Science (TUCS) (2012)
21. RODIN Modularisation Plug-in, `http://wiki.event-b.org/index.php/Modularisation_Plug-in` (accessed April 2012)
22. Sandvik, P., Neovius, M.: The Distance-Availability Weighted Piece Selection Method for BitTorrent: A BitTorrent Piece Selection Method for On-Demand Streaming. In: Liotta, A., Antonopoulos, N., Exarchakos, G., Hara, T. (eds.) Proceedings of The First International Conference on Advances in P2P Systems, AP2PS 2009, pp. 198–202. IEEE Computer Society (October 2009)

23. Sandvik, P., Sere, K.: Formal Analysis and Verification of Peer-to-Peer Node Behaviour. In: Liotta, A., Antonopoulos, N., Di Fatta, G., Hara, T., Vu, Q.H. (eds.) The Third International Conference on Advances in P2P Systems, AP2PS 2011, pp. 47–52. IARIA (November 2011)
24. Schulze, H., Mochalski, K.: Ipoque Internet Study (2008/2009), http://www.ipoque.com/en/resources/internet-studies (accessed April 2012)
25. Tarau, P.: Coordination and Concurrency in Multi-Engine Prolog. In: De Meuter, W., Roman, G.-C. (eds.) COORDINATION 2011. LNCS, vol. 6721, pp. 157–171. Springer, Heidelberg (2011)
26. Vestel to Launch the First Bittorrent Certified Smart TV, http://www.bittorrent.com/company/about/vestel_to_launch_the_first_bittorrent_certified_smart_tv (accessed April 2012)
27. Waldén, M., Sere, K.: Reasoning About Action Systems Using the B-Method. Formal Methods in Systems Design 13, 5–35 (1998)
28. Yan, L.: A Formal Architectural Model for Peer-to-Peer Systems. In: Shen, X., Yu, H., Buford, J., Akon, M. (eds.) Handbook of Peer-to-Peer Networking, Part 12, pp. 1295–1314. Springer, US (2010)
29. Yan, L., Ni, J.: Building a Formal Framework for Mobile Ad Hoc Computing. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2004, Part I. LNCS, vol. 3036, pp. 619–622. Springer, Heidelberg (2004)

Paper IV

A Formal Approach to H.264 Video Decoding on Multicore Systems

Kristian Lumme, Luigia Petre, Petter Sandvik, and Kaisa Sere

Originally published in *Int. J. Critical Computer-Based Systems*, Vol. 4, No. 1, pp. 3–26. Inderscience Publishers, 2013.
http://dx.doi.org/10.1504/IJCCBS.2013.053740

Partly based on the publication: Kristian Lumme, Luigia Petre, Petter Sandvik, and Kaisa Sere. Towards Dependable H.264 Video Decoding. In: Naveed Ahmed, Daniele Quercia, and Christian D. Jensen (Eds.), *Workshop Proceedings of the 5th IFIP WG 11.11 International Conference on Trust Management (IFIPTM 2011)*, pp. 325–337. Technical University of Denmark, June 2011.

A formal approach to H.264 video decoding on multicore systems

Kristian Lumme, Luigia Petre, Petter Sandvik* and Kaisa Sere

Department of Information Technologies,
Åbo Akademi University,
Joukahainengatan 3–5 A, 20520 Åbo, Finland
E-mail: kristian.lumme@abo.fi
E-mail: luigia.petre@abo.fi
E-mail: petter.sandvik@abo.fi
E-mail: kaisa.sere@abo.fi
*Corresponding author

Abstract: Multicore processing is quickly becoming ubiquitous, with more and more systems splitting the processing power between several processing cores instead of a single core. This approach is used not only to increase the total processing power and efficiency, but also to conserve energy. In this paper, we introduce a formal model created in Event-B of inter-frame dependencies in the H.264 video compression standard. Moreover, we formalise two parallelisation approaches for splitting the H.264 decoding process. The purpose of our modelling is to enable the adaptation of H.264 to multicore processing, as frames (or frame units called *blocks*) not depending on each other will be able to be decoded in parallel, on distinct cores. The formal proofs associated with the Event-B development of our model ensure the integrity of our proposal.

Keywords: video compression; H.264; multicore processing; formal method; Event-B; macroblock-level parallellisation; proving; refinement; RODIN-tool; frame-level parallelisation.

Reference to this paper should be made as follows: Lumme, K., Petre, L., Sandvik, P. and Sere, K. (2013) 'A formal approach to H.264 video decoding on multicore systems', *Int. J. Critical Computer-Based Systems*, Vol. 4, No. 1, pp.3–26.

Biographical notes: Kristian Lumme is currently studying for a Masters degree in Computer Science at Åbo Akademi University. Working as a research assistant at the Department of Information Technologies at this university, he has primarily focused on applying formal methods to a variety of fields, including multimedia and distributed computing.

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, meta-modelling. She has about 40 publications.

Petter Sandvik received his Masters degree in Computer Science from Åbo Akademi University in 2008. Currently, he is a PhD student at the Department of Information Technologies at the same university and working within the Distributed Systems Laboratory of Turku Centre for Computer Science (TUCS). His primary research interests are the formal aspects of media distribution systems, from the network structure to the media presentation.

Kaisa Sere is a Professor in Computer Science and Engineering at Åbo Akademi University since 1997 and 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 'Towards dependable H.264 video decoding' presented at the 5th Nordic Workshop on Dependability and Security (NODES 2011), Copenhagen, June 27–28, 2011.

1 Introduction

Computer systems with multiple processing cores are becoming commonplace. This increased interest in having more than one general-purpose processing unit may be explained with a renewed interest in efficiency, or power per watt, instead of absolute performance. Because energy requirements increase faster than performance gains when increasing clock frequencies, the efficiency can be greater when using multiple cores at lower frequencies than one single core at a higher frequency. However, while a computer system in general will benefit from being able to execute many things simultaneously, a single application cannot automatically use the additional processing power provided by multiple cores. Instead, a certain mapping of the process units to the multiple cores is needed.

H.264 is a video compression standard jointly developed by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) (ITU-T, 2010); compared to other video codecs it has very good visual quality per bit rate (Oelbaum et al., 2004). However, this quality comes at the expense of the processing power needed to decode video; H.264 is slower in decoding than its predecessors MPEG-4 and MPEG-2 (Alvarez et al., 2007). Therefore, there is interest in adapting H.264 to multiprocessing architectures, as described before (Meenderinck et al., 2008; van der Tol, 2003).

Mapping the H.264 video decoding process to multiple processing cores is in fact a very relevant problem, due to the ubiquity of video applications, from digital television to smart phones. Hence, we need to ensure the integrity of the decoding process. Formal methods, with their mathematical-based 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). In this paper we propose the use of formal methods for modelling and verifying the integrity of the H.264 decoding process on multicore systems. An important step towards this goal is that of identifying the H.264 units that can be processed on distinct cores, in parallel. By creating a formal model in which the dependencies between the units to be decoded are clearly defined, we can work towards a system in which units not depending on each other can be processed in parallel, thereby enabling the system to utilise multiple processing cores. We have chosen to use the Event-B language because of its successful use in real-life applications (for instance, see Craigen et al., 1994) and its excellent tool support (Abrial, 2007; Abrial et al., 2010), but our primary focus is not related to the usage of Event-B. Our goal in this paper is to describe a formal model of the dependencies among video frames and macroblocks to be decoded, thereby showing how the decoding of various independent units can be mapped to separate processing cores.

We envision this work as part of an envisioned complete model of a media distribution system, in which all the parts, from the acquisition of content to the actual media playback, have been formally specified, modelled and analysed. Parts of such a system have already been described elsewhere by us; for instance, Sandvik and Sere (2011) have modelled and analysed peer-to-peer content transfer algorithms for streaming media, and peer coordination has been modelled and described by Petre et al. (2012).

We proceed as follows. In Section 2, we discuss related work and in Section 3 we describe the Event-B formalism as used in this paper. Section 4 contains our description of the H.264 video compression standard. In Section 5 we put forward the development of our formal model of inter-frame dependencies, and in Section 6 we refine this model into models of two different parallelisation techniques. Section 7 contains discussion about our results, and we conclude this paper in Section 8 while also discussing future work.

2 Related work

Significant results are already documented in the literature on the theory of adapting H.264 decoding to multicore platforms. The two main ways of splitting the work load are *functional partitioning*, in which the different tasks of the decoding process are run on different processing cores, and *data partitioning*, where all processing cores do the same task but for different parts of the data. A comparison between these two main methods has been done by van der Tol (2003), and they note that data partitioning provides greater scalability. Baaklini et al. (2010) have explored a functional partitioning in which the luminance and colour channels are decoded in parallel. Within data partitioning, there are different solutions, and Meenderinck et al. (2008) have compared

frame-, slice- and macroblock-level parallelisation, later expanded to include partitioning into groups of pictures as well as different macroblock-level solutions.

One of the reasons for adapting video decoding to multicore platforms is to increase energy efficiency, and Kiliçarslan et al. (2011) have studied the power efficiency when increasing the number of processing threads, both for frame- and macroblock-level parallelisation. Seitner et al. (2011) have explored how to assign the macroblocks to distinct processor cores in order to provide optimal performance, especially for architectures with restricted resources. To achieve great scalability on different types of multicore systems, Chi and Juurlink (2011) have presented a quad high definition (QHD) capable parallel H.264 decoder that utilises both functional and data partitioning, as well as overlapping decoding of consecutive frames to avoid limited parallelism at the beginning and end of each frame.

An approach to mapping components to hardware platform tiles was presented in Petre et al. (2011a, 2011b). Components that displayed a high interaction level with each other were placed on close-by tiles in Petre et al. (2011a), to reduce the communication cost due to their dependencies. This was further elaborated in Petre et al. (2011b), where a detailed placing algorithm was introduced and placing was studied in both two- and three-dimensional networks-on-chip.

3 Event-B

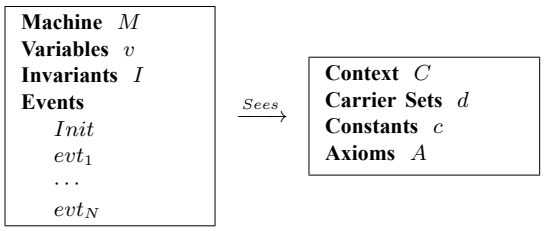
The B-method (Abrial, 1996) is a formal approach for the specification and development of highly dependable software. The method has been successfully used in the development of several complex real-life applications (Craigen et al., 1994; Event-B, n.d.). Event-B (Abrial, 2010) is a formalism derived from the B-method and the action systems (Back and Sere, 1996; Waldén and Sere, 1998) framework, to model and reason about parallel, distributed and reactive systems. Event-B has the associated RODIN platform (Abrial, 2007; Abrial et al., 2010; Event-B, n.d.) which provides automated tool support for modelling and verification by theorem proving.

3.1 Event-B language

In Event-B, a system specification (model) is defined using the notion of a *machine* (Event-B, n.d.) 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 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 Figure 1. The relationship between a machine and its accompanying context is called *sees* and denotes 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.

Figure 1 A machine M and a context C in Event-B



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:

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

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 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, $x := E(x, y)$, stands for assigning the value of the expression $E(x, y)$ to the variable x. 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 of x'. As a result of such a non-deterministic assignment, x can get any value belonging to Set or according to P.

3.2 Event-B semantics

The semantics of Event-B actions is defined using so-called before-after (BA) predicates (Abrial, 2010; Event-B, n.d.). A before-after predicate describes a relationship between the system states before and after execution of an event, as shown in Figure 2. In Figure 2, x and y are disjoint lists (partitions) of the state variables, and x', y' represent their values in the after-state. A before-after predicate for events is constructed as follows:

$$BA(evt) \;=\; g \wedge BA(S).$$

The semantics of a whole Event-B model is formulated as a number of *proof obligations*, expressed in the form of logical sequents. The full list of proof obligations can be found in Abrial (2010).

Figure 2 Before-after predicates

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

3.3 System development

Event-B employs a top-down refinement-based approach to system development. Development starts from an abstract system specification that models some of the essential functional requirements. When capturing more detailed requirements, a refinement step typically introduces new events and variables into the abstract specification. These new events correspond to stuttering steps that are not visible in the abstract specification. Old events can also be modified, typically either updating the newly introduced variables or introducing more deterministic assignments on the old variables, while also strengthening the event guards. This model refinement is referred to 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 the refined machine formally defines the relationship between the abstract and concrete variables; this type of invariants are called *gluing invariants*. Out of these refinement approaches, we employ the superposition refinement in this paper. To verify correctness of a refinement step, we need to prove a number of proof obligations for the refined model.

4 H.264 and parallelisation techniques

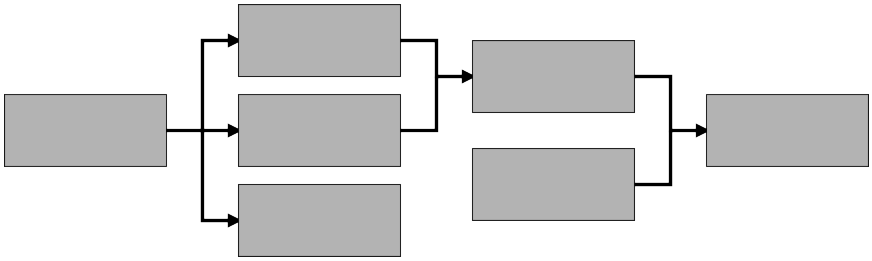
In H.264 (ITU-T, 2010), video frames are partitioned into *macroblocks*, which are blocks of typically 16 $\times$ 16 pixels (but may be as low as 4 $\times$ 4), upon which the discrete cosine transform (DCT), deblocking, and other processing occurs. A group of macroblocks in scan order (left to right, top to bottom) is called a slice, of which there is at least one in each frame. Slices can be of three types: I-slices, which are independent of other slices; P-slices, which depend on one or more previous slices; and B-slices, which depend on both past and future slices (Flierl and Girod, 2003). Having many slices per frame increases overhead (Garrett-Glaser, 2010) and decreases compression quality. In the common case, which we assume here, each slice therefore corresponds to a whole frame, which gives us I-, P- and B-frames, respectively.

When decoding an encoded video sequence, the order in which the frames are decoded (the processing order) does not necessarily correspond to the order in which they appear in the video sequence (the display order). There is often very little difference between two consecutive frames, or between two macroblocks in different frames, and therefore an encoder may choose to encode the difference between the current and a previously encoded one (along with a reference to the previous one), requiring less data than if all the data in a frame or macroblock had been encoded. This is referred to as *prediction*. In this case, when decoding the video sequence, the referenced frame has to be decoded before the one that references it, and because frames may depend on other frames both earlier and later in the display order, the processing order can

therefore be different from the display order. The choice between two frames without any dependencies on each other can be seen as non-deterministic.

Isović and Fohler (2002) found that the large majority of frames in MPEG-2 video streams are P- or B-frames. Less than ten percent of the frames in a typical sequence were found to be I-frames, and in our experience H.264 can go even further and sometimes consist of less than one percent I-frames. This shows the need for understanding dependency relations between frames, illustrated in Figure 3. We formally model these dependencies towards the goal of demonstrating how certain frames can be processed in parallel.

Figure 3 An example of a dependency relation between frames



5 Modelling dependencies

To begin with, we formalise the dependencies described earlier as a relation on a set of frames. We introduce an event that takes a single frame, eligible for decoding, and switches its state to indicate that it has been decoded. We then extend the model to reflect the fact that every frame is made up of many smaller blocks, the *macroblocks*, and that the prediction described above actually takes place on the macroblock level. The consistency between the frame-level dependency relation (which is really an abstraction) and the macroblock-level dependencies is ensured by invariants. In the following development steps, we also split the set of frames into two different ones, representing an input stream and an output stream. In Figure 6, we show an overview of the refinement process of our model.

5.1 Initial model

The concepts we model in the first version of the model are those of a set of (initially) unprocessed frames, an event that processes each of these frames one frame at a time, and some constraints on the processing order. The model can be interpreted as a distributed programme that could be run by several processing elements, decoding a video stream in parallel, along with some properties of the environment in which these processing elements function.

SETS

`F`

CONSTANTS

`dep E`

AXIOMS

$E \subseteq F$

$E \neq \varnothing$

$dep \in E \leftrightarrow E$

$\forall S \cdot S \subseteq E \wedge S \subseteq dep[S] \Rightarrow S = \varnothing$

The frames exist as a carrier set, F. For the purposes of the initial model we deal with a subset of F called E, representing encoded frames. The dependency relation, dep, is introduced as a binary constant relation on the set E. The last axiom states that no cycles are allowed in the inter-frame dependency relation.

VARIABLES

`proc`

INVARIANTS

$proc \in E \rightarrow BOOL$

$dep[proc^{-1}[\{TRUE\}]] \subseteq proc^{-1}[\{TRUE\}]$

Event *process* $\widehat{=}$

any

`x`

where

$x \in E$

$proc(x) = FALSE$

$dep[\{x\}] \subseteq proc^{-1}[\{TRUE\}]$

then

$proc(x) := TRUE$

end

The variable $proc$ keeps track of whether a frame has been processed or not. The second invariant above expresses the fact that if a frame has been processed, all of its dependencies must have been processed as well. A single event, $process$, takes any frame that is eligible for processing (that is, it has not been processed and if it has any dependencies, those have all been processed as well), and simply switches the state for that frame to true, indicating that it has been processed.

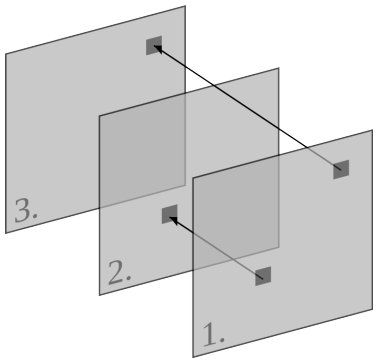
As mentioned in Section 3.1, in Event-B variables are initialised in the $Init$ event. However, because of space considerations the contents of this event are omitted here as well as in the further refinements.

5.2 First refinement

In the first refinement, a small detail is taken care of. As mentioned in Section 4, there are three basic types of frames in H.264, I-frames, P-frames and B-frames. The full meaning of these is explained in the later development steps. At this level, we only model the set $TYPES = \{I, P, B\}$ and the frame type as a constant $type : E \rightarrow TYPES$ and require that I-frames are not predicted from other frames. This requirement is modelled as the axiom $type^{-1}[\{I\}] \cap dom(dep) = \varnothing$. Hence, when decoding an I-frame it is not necessary to check if its dependencies have been processed, because the dependencies of I-frames are the empty set, and therefore we can modify the third guard of our $process$ event to specify that only frames other than I-frames have the requirement that their dependencies must have been processed.

Event *process* $\widehat{=}$
refines *process*
any
x
where
$x \in E$
$proc(x) = FALSE$
$type(x) \neq I \Rightarrow dep[\{x\}] \subseteq proc^{-1}[\{TRUE\}]$
then
$proc(x) := TRUE$
end

Figure 4 Inter-frame macroblock dependencies



Note: Due to the dependencies of its blocks, the decoding of frame 1 will depend on frames 2 and 3 having been decoded first.

5.3 Second refinement

We introduce more radical changes in the second refinement. As mentioned in Section 4, every frame consists of many *macroblocks*. The prediction process actually takes place

at the macroblock level. This also means that the dependencies between frames as related to the processing order are really dependencies between macroblocks. Generally, every macroblock in a frame is predicted either from neighbouring blocks in the same frame or from one or two macroblocks in other frames. In the former case, the blocks used for prediction must be blocks earlier in the processing sequence. This means that the dependencies between blocks in the same frame can be taken care of simply by processing the blocks in sequential order. In the latter case, the block-to-block dependencies naturally give rise to processing order constraints on their respective parent frames, so the dep relation from earlier can be seen as an abstraction of these block-to-block dependencies. We illustrate this in Figure 4. We can keep using this abstraction in the guards of events to ensure the constraints on the processing order are satisfied and, when the block-to-block dependencies are actually formalised, we ensure that these are consistent with the abstraction using axioms.

Here we also describe the idea of the I-, P- and B-frames. In an I-frame, every macroblock is predicted from another one in the same frame. In a P-frame, blocks can also be predicted from one block from another frame and in a B-frame, blocks can be predicted by a weighted average from blocks in two other frames.

CONSTANTS

`n`

AXIOMS

$n \in \mathbb{N}$

VARIABLES

`index processing processed`

INVARIANTS

$index \in E \rightarrow 1 \mathinner{..} n+1$

$processing \subseteq E$

$processed \subseteq E$

$processing \cap processed = \varnothing$

$\forall x \cdot x \in processed \cup processing \Rightarrow dep[\{x\}] \subseteq processed$

$processed = proc^{-1}[\{TRUE\}]$

$E \setminus processed = proc^{-1}[\{FALSE\}]$

While macroblocks are not yet explicitly formalised in the second refinement, the model now reflects the fact that a frame is not processed in one sweep, but incrementally, macroblock by macroblock. To accomplish this, the function $index$ gives every frame an index, which tells us how many of the macroblocks in that frame have been processed. The constant n represents the total number of macroblocks in a frame.

This also means that recording only whether a frame is processed or not is not enough anymore. We now need to keep track of which frames are currently undergoing processing as well. For this purpose, we introduce two subsets of E, $processed$ and $processing$. The set of unprocessed frames is E minus these two sets. As these sets are clearly related to the variable used to keep track of the state of frames before, $proc$, we add two so called *gluing invariants*. These describe the relationships between abstract and concrete variables and are shown above as the last two predicates.

Event *start* $\widehat{=}$
any
x
where
$x \in E$
$x \notin processing \cup processed$
$type(x) \neq I \Rightarrow dep[\{x\}] \subseteq processed$
then
$processing := processing \cup \{x\}$
end
Event *step* $\widehat{=}$
any
x
where
$x \in processing$
$index(x) \leq n$
then
$index(x) := index(x) + 1$
end
Event *stop* $\widehat{=}$
refines *process*
any
x
where
$x \in processing$
$index(x) = n + 1$
then
$processing := processing \setminus \{x\}$
$processed := processed \cup \{x\}$
end

All these developments require us to rethink the single $process$ event. More specifically, it has to be split into several events. We add the events $start$, $step$ and $stop$, of which the $stop$ event is set as a refinement of the $process$ event. What these events do is fairly self-explanatory: $start$ takes a frame eligible for processing and adds it to the set of frames being processed; $step$ takes one of the frames being processed and increments its index, thus representing the decoding of a single macroblock; and $stop$ simply takes a frame where the index of the frame is equal to the number of macroblocks in a frame plus one and sets its state to processed.

Setting the $stop$ event to be a refinement of the abstract event $process$ means we have to discharge some proof obligations to show that this really is the case. One of these is *guard strengthening*: we have to prove that the guards of the concrete event (along with the invariants and axioms of the model) imply the guards of the abstract one. In this case, one example is the proof obligation that the second guard of the abstract event, namely $proc(x) = FALSE$, is implied by the guards of the concrete event, along with the axioms and invariants of the model. Using the first guard of the concrete event, $x \in processing$, along with the gluing invariant mentioned earlier, $E \setminus processed = proc^{-1}[\{FALSE\}]$, this is easy to prove; however, without the gluing invariant it would have been impossible. In this manner, refinement can be used even as events and variables change.

5.4 Third refinement

We now add the macroblocks to the model as a carrier set, C, in the context. We then associate each frame with not simply an unordered set of macroblocks, but with an actual ordered sequence of them. This is accomplished through the constant $blockseq$ which is a total injection from the set of encoded frames to an ordered sequence of macroblocks; an ordered sequence of macroblocks is, in turn, a total injection from a sequence of natural numbers to the set C.

SETS

`C`

CONSTANTS

`sequence blockseq b_dep1 b_dep2`

AXIOMS

$sequence = 1 \mathinner{..} n \rightarrowtail C$

$blockseq \in E \rightarrowtail sequence$

$b\_dep1 \in C \nrightarrow E$

$b\_dep2 \in C \nrightarrow E$

$dom(b\_dep1) \subseteq ran(union(blockseq[type^{-1}[\{P, B\}]]))$

$dom(b\_dep2) \subseteq ran(union(blockseq[type^{-1}[\{B\}]]))$

$\forall x \cdot x \in E \wedge type(x) \neq I \Rightarrow dep[\{x\}] = (b\_dep1 \cup b\_dep2)[ran(blockseq(x))]$

For the macroblocks that depend on one macroblock in another frame, the function b_dep1 is introduced and for those which depend on two, the function b_dep2 is introduced. These are functions from macroblocks to frames, not to other macroblocks. While the latter is more consistent with the way these dependencies actually work, the only thing relevant to us at this point is the parent frame of the referenced macroblock, so instead of having the function point to another macroblock and then using an expression to get the parent frame of that block, we have the function point directly to the frame in question.

The last axiom above states that for any encoded frame, the dependencies of all its macroblocks make up the dependencies of the frame itself, that is, the macroblock level dependencies are consistent with the frame level abstraction.

VARIABLES

`b_proc`

INVARIANTS

$b\_proc \in C \rightarrow BOOL$

$\forall x \cdot x \in E \Rightarrow blockseq(x)[1 \mathinner{..} index(x) - 1] \subseteq b\_proc^{-1}[\{TRUE\}]$

$\forall x \cdot x \in processed \Rightarrow blockseq(x)[1 \mathinner{..} n] \subseteq b\_proc^{-1}[\{TRUE\}]$

Event $step \mathrel{\widehat{=}}$

refines *step*

any

`x`

where

$x \in processing$

$index(x) \leq n$

then

$b\_proc(blockseq(x)(index(x))) := TRUE$
$index(x) := index(x) + 1$

The variable b_proc tracks whether a certain macroblock has been processed or not. Invariants make sure that all the blocks in a frame with a lower sequence number than the current index of that frame have been processed and, subsequently, that in a processed frame, all the blocks have been processed. The $step$ event is refined to switch the state of the current block to indicate that it has been processed.

5.5 Fourth refinement

The fourth refinement refers to how we use the set of frames we are operating on. Until this point, we have modelled the subset of encoded frames the entire time, changing the state of the frames and blocks in place to represent decoding them. However, it is more realistic to have an input stream and an output stream, gradually building the output stream as the input stream is processed. This is now reflected in the model.

CONSTANTS

`block_cor encblocks`

AXIOMS

$encblocks = ran(union(blockseq[E]))$
$block\_cor \in encblocks \rightarrowtail C$

The output stream will have to be 'built up' as we process the input stream. Our earlier approach of changing the state of the elements of the video sequence does not translate well to this new approach. Instead, we now introduce processing based on a constant, $block\_cor$, which is a total injection giving each macroblock in the input stream $encblocks$ a corresponding one which is not in the input stream. This reflects the fact that for every encoded macroblock, decoding it will give us another macroblock, even though the former and the latter are represented quite differently on disk or in memory.

VARIABLES

`D frame_cor dblockseq`

INVARIANTS

$D \subseteq F \setminus E$
$frame\_cor \in (processing \cup processed) \rightarrowtail D$
$dblockseq \in D \rightarrow sequence$
$\forall x, y \cdot x \in D \wedge y \in D \wedge x \neq y \Rightarrow ran(dblockseq(x)) \cap ran(dblockseq(y)) = \varnothing$

Another subset of F, named D, is introduced, representing our output stream (as a set of decoded frames). Frames in the output stream are associated with macroblock sequences in a way analogous to frames in the input stream, but this time as a variable rather than a constant. This is because we model the decoding process, hence the encoded frames are given as a constant and the decoded frames are determined in the process and stored in a variable.

As the events can choose a frame to work on non-deterministically, process one block, and then pick another frame, we need a way to keep track of which frame in the

output stream corresponds to a specific frame in the input stream. This is modelled with the variable $frame\_cor$, which, illustrated below, is initialised for a specific frame by the $start$ event, as the frame in question is added to the set $processing$.

Event *start* $\widehat{=}$
refines *start*
any
`x y z`
where
$x \in E \setminus (processing \cup processed)$
$type(x) \neq I \Rightarrow dep[\{x\}] \subseteq processed$
$y \in F \setminus (E \cup D)$
$z \in sequence$
$z \notin dblockseq[D]$
then
$processing := processing \cup \{x\}$
$D := D \cup \{y\}$
$frame\_cor := frame\_cor \cup \{x \mapsto y\}$
$dblockseq(y) := z$
end

For the purposes of discharging various proof obligations, it is convenient to have $dblockseq$ be a total function from the set of decoded frames to a sequence of blocks. The event $start$ adds a frame y to the set D, but as this frame has not been processed yet, we do not have a block sequence for it. Still, for $dblockseq$ to be a total function, the frame needs to be assigned a sequence of blocks at this point. This is taken care of by adding a parameter $z \in 1 \mathinner{..} n \rightarrowtail C$ to the event and ensuring by various guards that the sequence is not one already 'in use' for any frame in D. This is just 'random initialisation data', resembling, in a way, the data that happens to be at a location in memory as we assign a pointer to that location in a programming language like C.

Event *step* $\widehat{=}$
refines *step*
any
`x y`
where
$x \in processing$
$index(x) \leq n$
$y = frame\_cor(x)$
then
$index(x) := index(x) + 1$
$dblockseq(y) := dblockseq(y) \Leftarrow \{index(x) \mapsto block\_cor(blockseq(x)(index(x)))\}$
$b\_proc(blockseq(x)(index(x))) := TRUE$
end

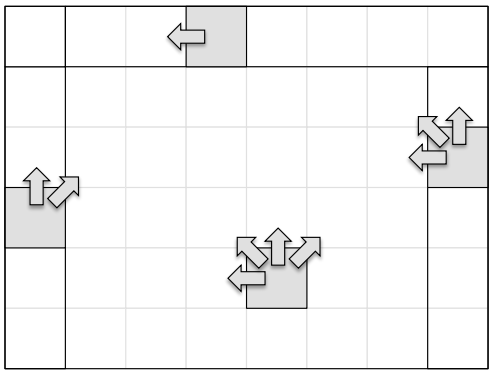
The $step$ event shows how a block is processed in this refinement. Recall, then, our current situation: the $step$ event is supposed to process a single macroblock of a single frame; we have an index telling us how far we are in processing the current frame; we also have a variable, $frame\_cor$, giving us the corresponding output frame of the input frame we are processing and a constant, $block\_cor$, giving us a (decoded) macroblock corresponding to the current (encoded) one. The main action of the $step$ event, then, is as shown above, namely it replaces whatever data is at the current index in the output frame with the macroblock corresponding to the one being processed.

6 Parallelisation models

Based on our model of inter-frame dependencies, we will now take a step towards parallel decoding by modelling how the decoding process can be mapped onto multiple processing units, such as processors or processor cores. One way of doing this is to use frame-level parallelisation, in which each frame to be processed is assigned to a specific processing unit. We model this in Section 6.1. Another way of parallelising decoding is to schedule different macroblocks within a frame onto different processing units, which requires us to take into account the dependencies that may exist among macroblocks of the same frame. We model this type of parallelisation in Section 6.2. An overview of the refinement process of our model can be seen in Figure 6.

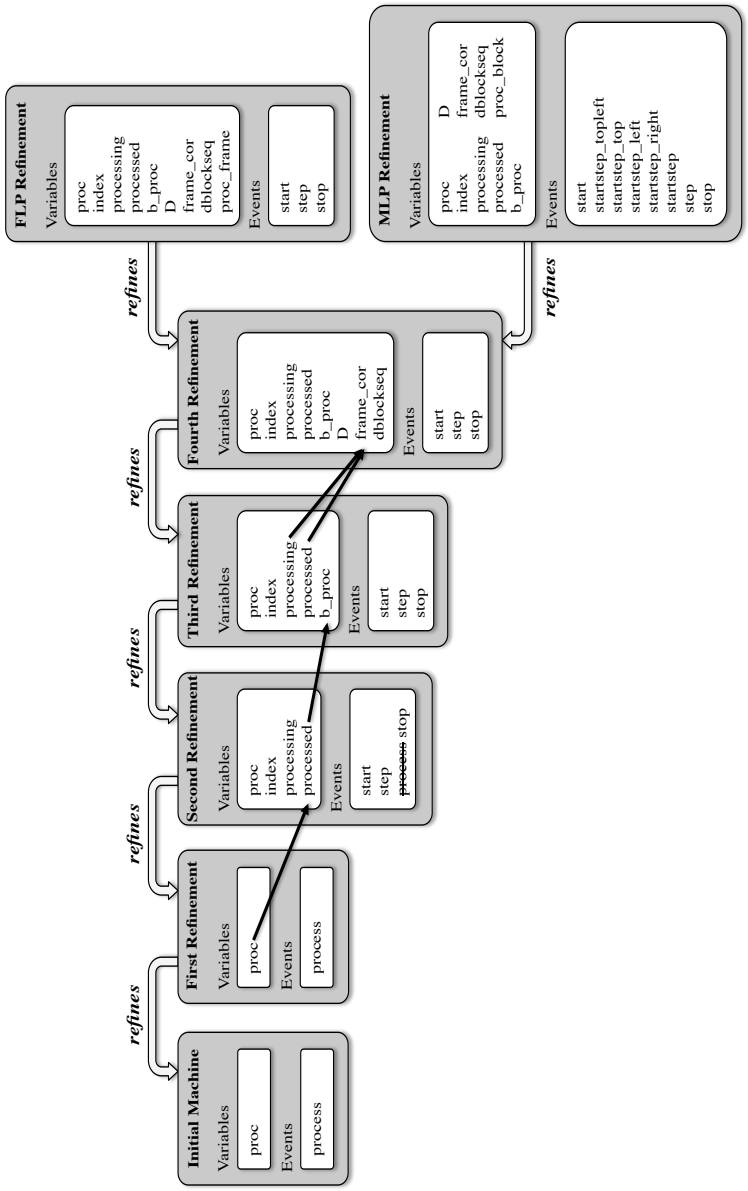
Because video decoding typically takes place on end-user devices where all processing units have access to the same memory, we have not considered the situation where non-uniform memory access, caches, or other hardware-specific details affect the parallelisation. If necessary, these could be introduced in further refinement steps.

Figure 5 Intra-frame macroblock prediction



Note: Each shaded block may be predicted from the blocks pointed to by its arrows, and those blocks must therefore be decoded before the shaded block.

Figure 6 Refinement of our model of H.264 inter-frame dependencies



Notes: After the fourth refinement step we separated our model into different refinements for frame-level and macroblock-level parallelisation. The arrows between variables indicate the existence of gluing invariants.

6.1 Frame-level parallelisation

In this refinement of our model, each frame being decoded is assigned to a specific processing unit. We assume that in our system all processing units share access to memory and data in such a way that we do not need to model this specifically, and therefore can provide an abstract representation of equal processing units.

SETS

`PU`

AXIOMS

$finite(PU)$

We begin by introducing a new finite set, PU, representing processing units. This is a constant, because we assume that the number of processing units will not change during execution. Even though in practice, frame-level parallelisation "limits the amount of TLP [thread-level parallelism] to a few threads" [Meenderinck et al., (2008), p.4], it is unnecessary for us to restrict our model further to a specific number.

VARIABLES

`proc_frame`

INVARIANTS

$proc\_frame \in E \nrightarrow PU$

$\forall f, p \cdot f \in E \land p \in PU \land (f \mapsto p) \in proc\_frame \Rightarrow f \in processing$

$\forall f \cdot f \in E \land f \in processing \Rightarrow (\exists p \cdot p \in PU \land (f \mapsto p) \in proc\_frame)$

In our model, only one frame at a time can be assigned to a specific processing unit, and likewise, a frame can only be assigned to one processing unit at a time. Hence, we add the variable $proc\_frame$ as a one-to-one relation from some frames to some processing units. The invariants specify that all frames that are assigned to any processing unit must be in $processing$, and that any frame in $processing$ must also have a processing unit assigned to it.

Event *start* $\widehat{=}$

refines *start*

any

`x y z v`

where

$x \in E \setminus (processing \cup processed)$

$type(x) \neq I \Rightarrow dep[\{x\}] \subseteq processed$

$y \in F \setminus (E \cup D)$

$z \in sequence$

$z \notin dblockseq[D]$

$v \in PU$

$v \notin ran(proc\_frame)$

$x \notin dom(proc\_frame)$

then

$processing := processing \cup \{x\}$

$D := D \cup \{y\}$

$frame\_cor := frame\_cor \cup \{x \mapsto y\}$
$dblockseq(y) := z$
$proc\_frame := proc\_frame \cup \{x \mapsto v\}$

end

Because each frame being processed must be assigned to a processing unit, we need to add a relation from the frame to the processing unit when we start processing a frame, i.e., in the $start$ event. For this to happen, we require that neither the frame nor the processing unit is already in use, meaning in the domain or range of $proc\_frame$, respectively.

Event *step* $\widehat{=}$
refines *step*
any
`x y v`
where
$x \in processing$
$index(x) \leq n$
$y = frame\_cor(x)$
$v \in PU$
$(x \mapsto v) \in proc\_frame$
then
$index(x) := index(x) + 1$
$dblockseq(y) := dblockseq(y) \lhd\!\!+ \{index(x) \mapsto block\_cor(blockseq(x)(index(x)))\}$
$b\_proc(blockseq(x)(index(x))) := TRUE$
end

The $step$ event needs to reflect the fact that each frame in $processing$ is assigned to a specific processing unit. Therefore, the $step$ event now takes an additional parameter specifying the processing unit, and the guards are strengthened by requiring that the frame being processed is assigned to that processing unit. The actions of the $step$ event do not need refining, as we still perform the same actions as previously.

Event *stop* $\widehat{=}$
refines *stop*
any
`x v`
where
$x \in processing$
$index(x) = n + 1$
$v \in PU$
$(x \mapsto v) \in proc\_frame$
then
$processing := processing \setminus \{x\}$
$processed := processed \cup \{x\}$
$proc\_frame := \{x\} \lhd\!\!- proc\_frame$
end

When we have finished processing a frame, we remove the relation between that frame and its assigned processing unit, in order to make the processing unit available to be assigned to another frame. This is done in the refined *stop* event by domain subtraction, which removes any relation with the specific frame as its domain.

6.2 Macroblock-level parallelisation

In addition to the frame-level parallelisation described in the previous subsection, we can also refine our model to specify macroblock-level parallelisation, in which the processing of each macroblock within each frame can be assigned to a specific processing unit. To do this, we must consider how macroblocks can be used for prediction within a frame, illustrated with Figure 5. In the general case, a macroblock can be predicted based on four other macroblocks: the ones immediately to the left, top left, top and top right. This constrains the order in which macroblocks can be processed, and we therefore need to specify how the macroblocks are positioned in the frame.

SETS

`PU`

CONSTANTS

`rows cols c_hpos c_vpos frame`

AXIOMS

$finite(PU)$

$rows \in \mathbb{N}_1 \setminus \{1\}$

$cols \in \mathbb{N}_1 \setminus \{1\}$

$rows * cols = n$

$c\_hpos \in C \rightarrow 1 \mathinner{..} cols$

$c\_vpos \in C \rightarrow 1 \mathinner{..} rows$

$frame \in C \nrightarrow E$

$dom(frame) = ran(union(blockseq[E]))$

$\forall x, c \cdot x \in E \wedge c \in ran(blockseq(x)) \Rightarrow frame(c) = x$

END

As in Section 6.1, we add a finite set representing the processing units, but here we also add constants specifying the number of rows and columns of macroblocks in each frame. The relations specifying the concrete horizontal and vertical position of a macroblock within a frame will be needed when determining the order in which the macroblocks can be processed. We also add the constant $frame$ as a more convenient way of relating a macroblock to the frame it is part of, forgoing the previous notation where we would need to refer to it using the sequence of blocks within a frame.

VARIABLES

`proc_block`

INVARIANTS

$proc\_block \in C \rightarrowtail PU$

The variable $proc\_block$ is a one-to-one relation from some macroblocks to some processing units, representing the macroblocks which are being processed. To model the actual processing, we introduce new events to start processing an eligible macroblock by assigning it to a specific processing unit, and then we refine the previous $step$ event to require that a macroblock be assigned a specific processing unit.

Event *startstep_top* $\widehat{=}$
any
`x v b b_left`
where
$x \in processing$
$index(x) \leq n$
$b \in C$
$v \in PU$
$v \notin ran(proc\_block)$
$b \notin dom(proc\_block)$
$blockseq(x)(index(x)) \in dom(block\_cor)$
$b = block\_cor(blockseq(x)(index(x)))$
$c\_hpos(b) \in 2 \mathinner{..} cols$
$c\_vpos(b) = 1$
$b\_left \in C$
$b\_left \in dom(frame)$
$frame(b\_left) = x$
$b\_proc(b\_left) = TRUE$
$c\_hpos(b\_left) = c\_hpos(b) - 1 \wedge c\_vpos(b\_left) = c\_vpos(b)$
then
$proc\_block := proc\_block \cup \{b \mapsto v\}$
end

As can be seen in Figure 5, besides the general case where each macroblock can be predicted from four other macroblocks, we also have four special cases where a smaller amount of macroblocks can be used for prediction, due to the macroblock being positioned near the edge of the frame. The first one of these is in the top left corner, which cannot be predicted from any other macroblock in the same frame, and this is therefore the first macroblock to be processed in a frame. The $startstep\_top$ event describes the case for the rest of the top row of macroblocks, where intra-frame prediction for each macroblock can only be based on the macroblock immediately to the left. Thus, for the unassigned macroblock b to be assigned to the unassigned processing unit v, we require that there is a processed block b_left in the same frame with the same vertical position but a horizontal position of one less. Similarly, we introduce events for processing the blocks in the leftmost column from the second row to the last, where macroblocks are predicted from top and top right; the rightmost column from the second row to the last, where macroblocks are predicted from left, top left and top; and the general case, used for most of the macroblocks in the frame.

The refined $step$ event, not shown here, simply requires that the macroblock given as a parameter must already have been assigned to a processing unit, and in that sense, the $step$ event now represents the act of finishing processing of a macroblock, including

removing the relation between that macroblock and the processing unit assigned to it. The ordered sequence of blocks referred to in Section 5.4 thereby does not necessarily refer to the display order, from top left to bottom right, but instead the order in which the macroblocks finishes processing.

7 Discussion

In this paper we formally develop the H.264 decoding model to a certain degree of detail, aiming towards suitable parallelisation techniques for the decoding process. In particular, we model the frame-level and the macroblock-level parallelisation techniques, starting from the same common formal model. The most important advantage put forward by our modelling is the possibility of providing mathematical proofs on the correctness of our models as well as on the stepwise derivation of our models. This aspect also underlies the power of abstraction in addressing model complexity, as employed by Event-B. Our initial model is very intuitive and simple, while our macroblock-level parallelisation is a rather complex model of the decoding process. However, we are certain of its correctness and various invariant properties, because we have derived it step-by-step from that initial simple model.

Our proposed modelling provides the basic step for more advanced applications related to modelling and analysing the H.264 compression standard. Thus, we can now model the actual processing taking place on the macroblocks, such as the DCT, deblocking, etc. When taking into account the particular topology of the processing units, other relevant properties can be modelled. For instance, we can assign particular processing units such as neighbour ones to decoding entities on whom other entities depend, so that communication among the latter and the former is optimised. Thus, knowing and modelling the locations of the processing units with respect to each other can lead to measuring the efficiency of the parallelisation technique. We can now also animate the models we have introduced. Based on tools such as ProB (Leuschel and Butler, 2003), we can for instance observe the execution traces of our two parallelisation proposals and compare them; an example of applying ProB to observe the execution traces was already shown in Petre et al. (2011a). Furthermore, we can refine our two final decoding models into a collection of smaller models to represent the distribution of the decoding on the processing units. This is possible based on a recent extension of Event-B, referred to as the modularisation extension (Iliasov et al., 2010). Via modularisation we can go a step further in modelling verifiable processes: we prove that a certain verified processing can be distributively implemented.

Finding a suitable level of abstraction also shows in the amount of interactive proving necessary for certifying the modelling. For our modelling, a total of 142 proof obligations were generated by the RODIN platform tool. Out of those, 125 were discharged automatically and 17 manually (interactively). In Table 1 we illustrate the distribution of these among the components of the model.

We note that the macroblock-level parallelisation was significantly more complex to model and prove than the frame-level parallelisation. This is due to two reasons. First, our most abstract model choice favours the frame manipulation, hence dealing with macroblocks adds to the model complexity. Second, the macroblock decoding involves the modelling of several smaller steps that also added to the complexity. However, our modelling is based on the RODIN platform tool and we can count the complexity in

terms of how many interactive proofs we get, instead of the total number of proofs. It is remarkable that we only got three proofs to interactively discharge for our final model, given the model size.

Table 1 Proof statistics

Component	*Total*	*Automatic*	*Interactive*
Contexts	4	3	1
Initial model	5	5	0
First refinement	2	1	1
Second refinement	24	20	4
Third refinement	16	11	5
Fourth refinement	19	16	3
Frame-level parallelisation refinement	9	9	0
Macroblock-level parallelisation refinement	63	60	3
Sum	142	125	17

8 Conclusions

In this paper we have introduced a formal model of the inter-frame dependencies in the H.264 video compression standard, as used in the decoding process. From the abstract level starting point, with a set of frames to be processed, we have refined our model into a more concrete one, with separate input and output streams of macroblocks. The purpose of doing this was to ensure the non-determinism of the order in which macroblocks belonging to frames not depending on each other can be processed, thus enabling these frames to be processed in parallel. We have then refined our model in separate directions, for frame-level parallelisation by mapping the processing of each frame onto a specific processing unit, and for macroblock-level parallelisation by doing a similar mapping for each macroblock within a frame. With these models on hand, we plan to continue by expanding our model in such a way that we could explore different macroblock-level parallelisation options. Another possible direction for our future work is to move towards the strategy described by Meenderinck et al. (2009) for combining frame- and macroblock-level parallelisation.

References

Abrial, J-R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F. and Voisin, L. (2010) 'Rodin: an open toolset for modelling and reasoning in Event-B', *International Journal on Software Tools for Technology Transfer (STTT)*, Vol. 12, No. 6, pp.447–466.

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

Abrial, J-R. (2007) 'A system development process with Event-B and the Rodin platform', in *Proceedings of International Conference on Formal Engineering Methods (ICFEM '07)*, Springer-Verlag, Vol. 4789 of *Lecture Notes in Computer Science*, pp.1–3.

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

Alvarez, M., Salamí, E., Ramírez, A. and Valero, M. (2007) 'HD-VideoBench. A benchmark for evaluating high definition digital video applications', in *IISWC '07 Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization*.

Baaklini, E., Sbeity, H., Niar, S. and Amaneddine, N. (2010) 'H.264 color components video decoding parallelization on multi-core processors', in *13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*.

Back, R.J.R. and Sere, K. (1996) 'From action systems to modular systems', *Software – Concepts and Tools*, Vol. 17, No. 1, pp.26–39.

Chi, C. and Juurlink, B. (2011) 'A QHD-capable parallel H.264 decoder', in *Proceedings of the International Conference on Supercomputing*, pp.317–326.

Craigen, D., Gerhart, S. and Ralson, T. (1994) 'Case study: Paris metro signaling system', in *Proceedings of IEEE Software*, pp.32–35, IEEE.

Event-B. (n.d.) *Event-B and the Rodin Platform* [online] http://www.event-b.org/ (accessed October 2012).

Flierl, M. and Girod, B. (2003) 'Generalized B pictures and the draft H.264/AVC video-compression standard', *IEEE Transactions on Circuits and Systems for Video Technology*, July, Vol. 13, No. 7, pp.587–597.

Garrett-Glaser, J. (2010) 'Threads in x264' [online] http://git.videolan.org/gitweb.cgi?p=x264.git;a=blob;f=doc/threads.txt (accessed October 2012).

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 M. Frappier, U. Glosser, S. Khurshid, R. Laleau and S. Reeves (Eds.): *Abstract State Machines, Alloy, B and Z*, Vol. 5977 of *Lecture Notes in Computer Science*, pp.174–188, Springer, Berlin/Heidelberg.

Isović, D. and Fohler, G. (2002) 'Analysis of MPEG-2 video streams', Technical report, Mälardalen Real-Time Research Centre, Mälardalen University, August.

ITU-T (2010) 'H.264: advanced video coding for generic audiovisual services' [online] http://www.itu.int/rec/T-REC-H.264-201003-S/en (accessed October 2012).

Kiliçarslan, D., Gürler, C.G., Özkasap, Ö. and Tekalp, A.M. (2011) 'Energy efficient video decoding on multi-core devices', in *e-Energy '11*.

Leuschel, M. and Butler, M. (2003) 'ProB: a model checker for B', in K. Araki, S. Gnesi and D. Mandrioli (Eds.): *FME 2003: Formal Methods*, Vol. 2805 of *Lecture Notes in Computer Science*, pp.855–874, Springer, Berlin/Heidelberg.

Meenderinck, C., Azevedo, A., Alvarez, M., Juurlink, B. and Ramirez, A. (2008) 'Parallel scalability of H.264', in *Workshop on Programmability Issues for Multi-Core Computers (MULTIPROG)*.

Meenderinck, C., Azevedo, A., Juurlink, B., Alvarez, M. and Ramirez, A. (2009) 'Parallel scalability of video decoders', *Journal of Signal Processing Systems*, Vol. 57, No. 2, pp.173–194.

Oelbaum, T., Baroncini, V., Tan, T.K. and Fenimore, C. (2004) 'Subjective quality assessment of the emerging AVC/H.264 video coding standard', in *Int. Broadcast Conference (IBC)*.

Petre, L., Sere, K. and Tsiopoulos, L. (2011a) 'Model-based analysis tools for component synthesis', in B.K. Aichernig, F.S. de Boer and M.M. Bonsangue (Eds.): *Formal Methods for Components and Objects, 9th International Symposium, FMCO 2010*, Vol. 6957 of *Lecture Notes in Computer Science*, pp.102–121.

Petre, L., Sere, K., Tsiopoulos, L., Liljeberg, P. and Plosila, J. (2011b) 'Towards self-placing applications on 2D- and 3D-NoCs', in P. Cong-Vinh (Ed.): *Autonomic Networking-on-Chip: Bio-inspired Specification, Development, and Verification*, pp.165–187, Embedded Multi-core Systems (EMS) Book Series, CRC Press.

Petre, L., Sandvik, P. and Sere, K. (2012) 'Node coordination in peer-to-peer networks', in M. Sirjani (Ed.): *Proceedings of the 14th International Conference on Coordination Models and Languages (COORDINATION 2012)*, Vol. 7274 of *Lecture Notes in Computer Science*, pp.196–211, Springer.

Sandvik, P. and Sere, K. (2011) 'Formal analysis and verification of peer-to-peer node behaviour', in *Proceedings of the Third International Conference on Advances in P2P Systems (AP2PS '11)*, pp.47–52.

Seitner, F., Bleyer, M., Gelautz, M. and Beuschel, R. (2011) 'Evaluation of data-parallel H.264 decoding approaches for strongly resource-restricted architectures', *Multimedia Tools and Applications*, Vol. 53, No. 2, pp.431–457.

van der Tol, E.B., Jaspers, E.G.T. and Gelderblom, R.H. (2003) 'Mapping of H.264 decoding on a multiprocessor architecture', in Vasudev, B., Hsing, T.R., Tescher, A.G. and Ebrahimi, T. (Eds.): *Image and Video Communications and Processing*, Volume 5022 of Proceedings of the SPIE, pp.707–718.

Waldén, M. and Sere, K. (1998) 'Reasoning about action systems using the B-method', *Formal Methods in Systems Design*, Vol. 13, No. 1, pp.5–35.

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

Paper V

Translation and Validation of SPECTA – A Specification Language for Content Transfer Algorithms

Petter Sandvik

Shorter version published as SPECTA: A Formal Specification Language for Content Transfer Algorithms. In *IEEE Fifteenth International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2014)*, pp. 638–641. IEEE, June 2014.

Translation and Validation of SPECTA – A Specification Language for Content Transfer Algorithms

Petter Sandvik
Faculty of Science and Engineering, Åbo Akademi University
TUCS – Turku Centre for Computer Science
Joukahaisenkatu 3-5A
20520 Turku, Finland
petter.sandvik@abo.fi

ABSTRACT

In centralised network systems based on the client-server paradigm, content transfer between nodes usually happens in-order. However, in more distributed systems, such as cloud-based systems and peer-to-peer networks, it can be advantageous to transfer data out-of-order. Transfer speed, availability and reliability can be improved through the use of different algorithms that take the distributed nature of the network into account. For the purpose of enabling easier understanding of the complexities of out-of-order transfer, the Specification for Content Transfer Algorithms (SPECTA) language has been proposed. In this article, we show how algorithms written in the SPECTA language can be automatically translated into other formalisms, such as Event-B, for the purpose of analysis, verification and code generation, as well as compare the formal models based on translated algorithms to previously created models.

1. INTRODUCTION

When information needs to be transferred over a network, the simplest way is by sending it from one node to one other node. This is usually referred to as the client-server model. However, it has long been recognised that there are situations where this one-to-one communication is not optimal, and a many-to-one or many-to-many type of communication would be preferable. Over the years, many names have been used to describe these types of networks; it has been popular to talk about them not only as distributed systems but also as peer-to-peer networks and cloud-based systems. However, the basic principle has remained the same: the advantage of having multiple nodes involved can be used to improve speed, reliability and availability.

In systems using the traditional client-server model, the data transfer usually happens in-order, i.e., the data is sent from the server in the same order it will be used on the client. Often, there is no need to even consider doing it any other way. However, in distributed systems, where multiple nodes are involved, there may be an advantage to use out-of-order content transfer algorithms. Utilising these algorithms, different parts of a particular content may be transferred to or from separate nodes in a different order than they originally were in, only to be reassembled in the correct order at the end. An example of such an approach is BitTorrent [11]. The advantages of these algorithms apply not only when the end consumer is involved, but also in other circumstances, such as when data is replicated in data centres. For instance, it has been reported that YouTube uses BitTorrent technology to replicate content within its data centres, and is "very pleased with the performance" [10].

Traditionally, these content transfer algorithms have often been described in a plain text format, intended for human interpretation. Other times, the way of describing them has been by actual program code, or in a formalism designed to be machine interpreted. The non-formal descriptions include BiToS [29] and DAW [24], and there has been work on more formal descriptions of these as well [26, 27]. Generally, when algorithms are described, especially for the purpose of learning, it is often done using pseudocode. This is a way of informally describing what an algorithm does, in a way that resembles code in a programming language. Unfortunately, pseudocode is not standardised (although there have been attempts at doing so [13]), and, due to the way program flow is structured differently in different languages, a particular example of pseudocode tends to be somewhat tied to a specific programming language or method.

For the purpose of comparison and evaluation of content transfer algorithms, as well as their further development, there is a need to have a reusable formal way of specifying these algorithms. This formalism should be powerful enough to support algorithms from such a wide variety of different distributed content transfer systems as possible, and we would also like to be able to use this formalism for automatic translation into other languages. This would be done not only for the purposes of modelling, verification and code generation, but also in order to take advantage of existing tool support for these purposes. One proposal for such a formalism is the Specification for Content Transfer Algorithms (SPECTA) language [22, 23].

This article is organised as follows. In Section 2 we describe

the SPECTA language itself to the extent needed in this article, as well as show a few examples of algorithms and their representation in SPECTA. In Section 3 we describe our ideas for translation from SPECTA to other languages, as well as two examples of translation into Event-B. Section 4 consists of related work. We conclude this article in Section 5 with discussion about our results and future work.

2. THE SPECTA LANGUAGE

We will here give a brief overview of the SPECTA language. More details can be found in the original sources [22, 23].

For out-of-order content transfer, it is assumed that the content to be transferred is partitioned into a finite number of sequentially numbered pieces, represented by positive integers starting from 1, and that the content transfer algorithms stepwise choose which eligible piece of content to transfer next, based not only on the properties of each eligible piece but also of the system as a whole. This should go on until there are no eligible pieces left to transfer. Deciding about which pieces are eligible at any given time is application-specific, and may even change over time, and thus is not contained in the algorithm.

In SPECTA, two different types of properties are used when an algorithm selects the next piece of content to be transferred. The *condition* specifies a state the system must be in for the selection to take place at all, and the *criteria* describe properties that a specific piece must have in order to get selected. The condition is separated from the criteria with a right triangle symbol ($\triangleright$), and the criteria from each other with a pipe symbol ($|$). The first criterion is used to specify a subset of all eligible pieces, the second criterion is used to specify a subset of the first subset, and so on. As we are only interested in choosing one piece at a time, it is assumed that if the final subset, as specified by the criteria, consists of more than one piece, one piece is non-deterministically chosen from that set. The piece selection in SPECTA can be seen in (1).

$$\begin{aligned} selection \equiv condition \ \triangleright \ & criterion_1 \\ | \ & criterion_2 \\ | \ & \dots \\ | \ & criterion_m, \text{ where } m \geq 1 \end{aligned} \tag{1}$$

In cases where the condition is not fulfilled, or the condition is fulfilled but no eligible pieces satisfy all criteria, no selection can take place according to (1). To avoid these situations, the semicolon can be used to combine multiple selection mechanisms (2). The semicolon is usually taken to mean sequential composition of statements, and in this case its usage is similar: if the first selection can take place according to the first selection mechanism, it does, but if not, the second selection mechanisms is tried, and so forth. However, unlike the situation with the criteria in (1), non-determinism is not involved here, but instead it is assumed that the algorithm has been designed in such a way that all possible ways of selecting a piece have been taken into account, and if the final selection mechanism fails to select a piece, the correct behaviour is to do nothing until the properties of the system and pieces have changed in such a way that the conditions and criteria make piece selection possible.

$$\begin{aligned} next \ = \ & selection_1; \\ & selection_2; \\ & \dots; \\ & selection_k, \text{ where } k \geq 1 \end{aligned} \tag{2}$$

For expressing the conditions and criteria, simple arithmetic operations are used. However, together with them a few specific keywords and operations are also needed. We show a few of the more important ones in Table 1. It should be noted that several of them are based on the notion of having pieces as sets, which are a more abstract representation than arrays or other data structures more commonly used in practice.

2.1 SPECTA Examples

Based on the keywords and operations of Table 1, we can now give a few examples of SPECTA. More of these can be found in [23].

The original BitTorrent peer-to-peer file sharing application [11] used, for the most part, a two-stage piece selection method. First, pieces should be requested randomly until one complete piece has been transferred. After one complete piece has been transferred, pieces should be selected rarest-first, i.e., lowest availability has highest priority. As the behaviour when more than one piece has the same, lowest availability is not specified [11], we should not include any criteria that restricts the selection to just one piece. Instead, we leave the choice non-deterministic when there is more than one piece in the subset with lowest availability (3).

$$\begin{aligned} next = \ & transferred < 1 \triangleright random(pieces); \\ & transferred \geq 1 \triangleright min(avail(piece)) \end{aligned} \tag{3}$$

While the original BitTorrent protocol found use in transferring files, it was not well suited for streaming of media content. However, there have been several proposals on how it could be modified to function better in such situations. One such modification is BiToS [29]. Another piece selection method for on-demand streaming media is the Distance-Availability Weighted method (DAW) [24, 25, 26]. In DAW, pieces are first requested sequentially from the current media playback position up to a certain buffer size. When all pieces belonging to this buffer have been requested, pieces outside the buffer are requested based on their priority. The priority for a piece is based on the distance from a piece to the last piece in the buffer, i.e., the number of pieces in between when they are ordered, multiplied by the availability of that piece. Thus, pieces that have low availability and are close to the content playback position have high priority, while pieces that have high availability and are far from the playback position will have low priority. As in BiToS, in the situation where more than one eligible piece has the same priority, the piece that has the lowest piece number is chosen. In SPECTA, the DAW algorithm can be written as (4). For this to work, any pieces with no availability would have to be excluded from the set of eligible pieces, but in practice this would be done anyway as there is no point in requesting pieces that are not

Table 1: Keywords and operations in SPECTA [22, 23].

Main form	Alternative form(s)	Description
`piece`	`p`	The ID of the specific piece being considered.
`total`	`all, last`	The total number of pieces. Also the ID of the last piece, because pieces are numbered from 1.
`eligible`	`elig`	The set of all eligible pieces, a subset of all pieces.
`pieces`		The subset of eligible pieces satisfying all criteria so far. For the first criterion, this is the same as `eligible`.
`requested`	`r, req`	The number of pieces that have been requested.
`transferred`	`t, tr, transfered`	The number of pieces that have been transferred. This number may be smaller than `requested` when transferring content in parallel.
`availability(x)`	`av(x), avail(x)`	The number of nodes that hold the piece of content specified by the parameter. If no parameter, `piece` is assumed.
`minimum(x)`	`min(x)`	The piece (in `pieces`) for which the parameter is the smallest.
`maximum(x)`	`max(x)`	The piece (in `pieces`) for which the parameter is the largest.
`random(x)`	`random(x,y)`	If the parameter is a set of pieces, returns a random piece from that set. Otherwise gives a random number from `1` to `x`, or with two parameters, in the range from `x` to `y`.
`probability(r)`	`prob(r)`	Returns `true` with the probability `r`, $0 \leq r \leq 1$.
`size(x)`		Returns the size of the piece specified by the parameter.
`current`	`c, cur`	The current piece in any external use, e.g., playback position in media streaming.

available and therefore cannot be transferred.

$$\begin{aligned} next = true \ \triangleright \ & piece \leq current + buffersize \\ & \mid min(piece); \\ true \ \triangleright \ & min(avail(piece) \\ & * (piece - (current + buffersize))) \\ & \mid min(piece) \end{aligned} \tag{4}$$

3. TRANSLATION FROM SPECTA

One idea behind a language such as SPECTA is that it can enable translation from a human-readable format into a language aimed at machine interpretation. Depending on the use case, this language could for instance be a programming language like C or Java. However, in our work we started by translating into the the formal modelling language Event-B [2]. We have two major reasons for this. Firstly, there is excellent tool support for Event-B in the form of the extensible Rodin Platform tool [14]. This supports plug-ins for a wide variety of use cases, such as animation and model checking [19] as well as code generation [15, 20, 31]. Secondly, the existence of previous work with content transfer algorithms in Event-B [26, 27] facilitates easier comparison between models based on automatically translated algorithms and those that previously modelled using Event-B. In the following, will first describe Event-B and secondly present our approach to translation from SPECTA to Event-B and show examples of translated algorithms.

3.1 The Event-B Language

The B-Method [1] is a formal approach to specifying and developing highly dependable software, used successfully in development of several complex real-life applications [12, 14]. From the B-Method and the Action Systems [5, 7, 30] framework Event-B [2, 14] was derived for the purpose of modelling and reasoning about parallel, distributed and reactive systems. The previously mentioned Rodin Platform [3, 14] was developed to offer tool support, which is an important asset for facilitating widespread adoption.

In Event-B, a model of a system is made up of two parts: a *machine*, usually referred to as the dynamic part of the model, and a *context*, seen as the static part of a model. Technically, an Event-B context is optional in a model, although in practice one is always included, as the context specifies constants, carrier sets, and axioms about these to be used in the model. An Event-B machine *sees* a specific context, and holds the model state in *variables*, which are updated by *events*. An event is an atomic set of variable updates, happening simultaneously, and each event may also contain *guards*. The guards of an event are associated predicates that must evaluate to true for the event to be able to execute, i.e., be *enabled*. If more than one event is enabled simultaneously, the choice between the events is non-deterministic. An Event-B machine should also include *invariants*, which are properties that must hold for any reachable state of the model. Thus, the properties specified by the invariants must hold before and after each occurrence of any event, after having been established by the **INITIALISATION** event. To be able to prove that this happens, the proof manager in the Rodin Platform tool automatically generates what needs to be proved in order for an invariant to hold.

Event-B provides a stepwise refinement-based approach to system development, where correctness is preserved by gradually introducing new variables, events and constants in a manner that does not disturb the previous functionality. Refinement starts from an abstract model, which describes what the system should do, and with each refinement step the system becomes more concrete, describing how it should do what it was designed to do. Refinement can be *horizontal* or *vertical*, where horizontal or superposition refinement [8, 17] refers to adding new variables, events and constants in addition to existing ones. Previous events can also be modified, typically such that they either update the newly introduced variables or introduce more deterministic assignments on the pre-existing variables, while also strengthening the event guards. Vertical refinement, or data refinement [6], corresponds to replacing some abstract variables with their concrete counterparts and accordingly changing the events. We do not use refinement when translating from SPECTA, although it is essential when further modelling in Event-B, and it must be noted that our translation fits in somewhere between the most abstract model and the most concrete one. For instance, the algorithms written in SPECTA will give the number of the piece of content that should be requested

for transfer, which can be seen as a refinement of an abstract event specifying that the "optimal" piece should be found. Likewise, in SPECTA we can specify the minimum or maximum of something within a set, which, conveniently, is a built-in function in Event-B, but a more concrete implementation would specify how the minimum or maximum should be found. In the following, we describe some of the other details that need to be considered when translating SPECTA into Event-B.

3.2 Translating SPECTA to Event-B

Although SPECTA is different from Event-B, they are both, to a certain extent, text-based formats. Therefore, we decided to start by using a scripting language with good text processing abilities for translation. Based on that, in combination with our preference to use a platform-independent language, we chose to write our simple SPECTA to Event-B translator in Perl. It must be noted that we do not translate SPECTA into Event-B models, nor into Event-B machines, but into Event-B code representing events. For this reason, the output of our translator cannot directly be used in the RODIN Platform tool without adding variables and other events to the Event-B machine, as well as relevant constants and axioms to the Event-B context. We have made a deliberate decision to do it this way, because simply translating does nothing except show that translation is possible. Any practical use of the translated code in Event-B models should include additions of invariants as well as other events in such a way that there are real properties that should and could be proved.

When compared to SPECTA, Event-B has certain limitations. For instance, there is no support for random numbers or probabilities. In the former case we have opted to translate to the built-in non-determinism in Event-B, and for the latter we have translated such that we have a Boolean variable *probability* and an event that non-deterministically changes its value to true or false. This is because both randomness and probabilities can be seen as special cases of non-determinism; that is, anything we can prove about something that is true in a random way or with a certain probability we must be able to prove regardless of whether it is true or not. In other words, we now abstract probabilities and randomness into non-determinism, and assume that when or if Event-B can support these concepts natively we can reintroduce them as refinements of this non-determinism. Another problem we faced when translating is that although we assume content pieces are numbered using integer numbers, SPECTA can work with real numbers in other places, while Event-B currently only supports integers. Together, these limitations mean that few algorithms written in SPECTA could be successfully translated completely without human intervention, although in most cases it will be possible to rewrite the algorithms in such a way that they can be translated, at least for the purpose of including them into a bigger model for the purpose of proving.

The first approach we tried for translating SPECTA to Event-B was a naïve implementation, in which each of the selections combined was translated into a single event. Initially, the first event would be enabled, and the second event would be enabled if the first one selection failed, and so forth. This worked well for simple algorithms, such as ones consisting of only a few selections, each containing no more than two criteria. However, because each event had to specify all the conditions under which it would be enabled, and therefore needed as its guard all the possible ways the previous events could fail to select a piece, this approach became unmanageable for larger algorithms.

The second approach we tried involved splitting the events from the first approach into several ones. Instead of a single event for each selection, we had multiple events based on the different ways the previous selection could fail to select a piece. Thus, the first selection had one event, and the second selection had one event enabled when the first selection was unsuccessful because of the condition, and one additional event for each criteria that could cause the first selection to be unsuccessful. The third selection would then increase the number of events even further. The large number of events decreased the readability of the generated Event-B code and also increased the time needed to prove anything, as proving that an invariant holds needs to be done separately for each event. Therefore, it soon became obvious that this approach was not ideal either.

The third approach we tried, which is also the one we will describe here and show an example of, is based on the fact that the second selection onwards only need to know that the previous selections were unable to select a piece, not the reason why they were unable to do so. We could then use a variable specifying which selection we were considering at the moment, knowing that if we considered selection method number two, selection method number one must for some reason have been unable to select a piece. This relationship could even be defined formally as an invariant stating that if we were considering the second selection, either the condition for the first selection must have been unfulfilled, or there was no eligible piece satisfying all criteria. In fact, this invariant must hold for all selections after the first one, and a similar invariant could be made regarding each selection method and made to hold for all subsequent ones.

A representation of program flow within the Event-B events created from SPECTA code can be seen in Figure 1. For each selection, we have two events representing whether the condition is fulfilled or not (for instance, **SP_SELECT_0** and **SP_SELECT_0_NEG**). To facilitate the use of multiple criteria in one selection, we also create a separate event for each criteria within each selection (**SP_SELECT_0_0**, **SP_SELECT_0_1**, ...). Each of these events defines *pieces* to be a new subset of the previous *pieces*, exactly as described in Section 2, but we also add one final event that, if *pieces* is non-empty, non-deterministically selects a piece from that set (**SP_SELECT_0_COMPLETE**). If the set *pieces* is empty at any time, meaning that one of the criteria is such that no eligible piece fulfills that one and all the previous ones, we reset the set *pieces* to contain all eligible pieces (in **SP_SELECT_EMPTY**), and move to the next selection. Likewise, if the final selection is not successful in selecting a piece, the event **SP_SELECT_FAILED** sets *pieces* to all eligible pieces and lets the program flow within the SPECTA-created events return to the idle state.

As mentioned previously, the events generated from SPECTA code do not constitute a complete model. For instance, we

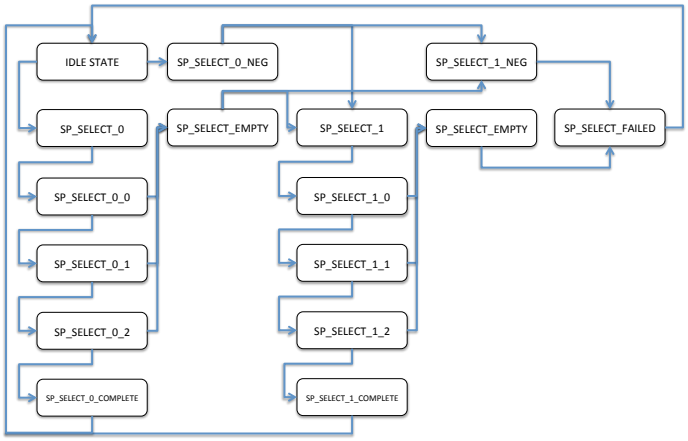


Figure 1: A representation of program flow within Event-B events created from SPECTA code, consisting of two selection methods each containing three criteria.

assume that the set of all pieces is constant, and therefore the number of pieces is also constant, and thus we can add these as constants in the context of our Event-B model. Likewise, *pieces*, *eligible*, *availability* and *next* are used by the events but not constant, and therefore we add them as variables to the Event-B machine and initialise them in the **INITIALISATION** event. We also have variables that model the program flow: *selection_method* describes which selection we are considering at the moment, roughly corresponding to the first digit or horizontal movement in Figure 1; *selection_step* describes which criterion we are looking at, roughly corresponding to the second digit or vertical movement in Figure 1; and *selection_inprogress* is a Boolean value representing whether we are doing piece selection, that is, whether we are outside the idle state in Figure 1 or not. The latter of these is done mainly to facilitate easier integration with events not generated from SPECTA code; if we need to prove anything about what happens in the piece selection events we need a way for other events updating variables such as *eligible* or *availability* to be prevented from doing so while piece selection is underway.

Because of space considerations, we can only show one example of SPECTA translated to Event-B events. Although we have also translated the BitTorrent piece selection method (3), the piece selection algorithm we show translated is DAW (4), previously presented in SPECTA form in Section 2.1. The first two events from DAW are shown in Figure 2. Here, we note two things. Firstly, in Event-B there is a difference between logical true ($\top$) and the Boolean value true (TRUE). The first one is used in the guards translated from SPECTA conditions, while the second one is a value that a variable can have. Secondly, as the condition here is $\top$ the event with the negation as guard is superfluous, since $\neg(\top)$ can never be true. However, for the sake of completeness the event is still generated and shown here.

event SP_SELECT_0
  where
    @grd1 $\top$
    @grd2 selection_method = 0
    @grd3 selection_step = 0
    @grd4 selection_inprogress = FALSE
  then
    @act1 selection_step := 1
    @act2 selection_inprogress := TRUE
end

event SP_SELECT_0_NEG
  where
    @grd1 $\neg(\top)$
    @grd2 selection_method = 0
    @grd3 selection_step = 0
    @grd4 selection_inprogress = FALSE
  then
    @act1 selection_method := 1
    @act2 selection_step := 0
    @act3 selection_inprogress := TRUE
end

Figure 2: Events corresponding to the condition of the first selection in DAW, and its negation.

Figure 3 shows the events corresponding to the criteria of the first selection in DAW. We first limit *pieces* to the pieces from *current* to *current* + *buffersize* (**SP_SELECT_0_0**), then we choose the piece from that set with the smallest piece number (**SP_SELECT_0_1**) and finally we choose one piece from the remaining subset (**SP_SELECT_0_COMPLETE**).

In Figure 4 we show the event corresponding to the condition of the second selection in DAW, and as in Figure 2 we also have a superfluous event which is only enabled when $\neg(\top)$ is true, i.e., never. However, we must point out that the reason this event can never be enabled here is because of the special case that the condition for this selection method is *true* and therefore never can be false.

Figure 5 shows the events generated based on the criteria of

```
event SP_SELECT_0_0
  any newpieces
  where
    @grd1 newpieces = { piece | piece ∈ pieces
            ∧ piece < current + buffersize }
    @grd2 selection_method = 0
    @grd3 selection_step = 1
    @grd4 selection_inprogress = TRUE
    @grd5 pieces ≠ ∅
  then
    @act1 pieces := newpieces
    @act2 selection_step := 2
end

event SP_SELECT_0_1
  any newpieces
  where
    @grd1 newpieces = { piece | piece ∈ pieces
            ∧ piece = min(pieces) }
    @grd2 selection_method = 0
    @grd3 selection_step = 2
    @grd4 selection_inprogress = TRUE
    @grd5 pieces ≠ ∅
  then
    @act1 pieces := newpieces
    @act2 selection_step := 3
end

event SP_SELECT_0_COMPLETE
  where
    @grd1 selection_method = 0
    @grd2 selection_step = 3
    @grd3 selection_inprogress = TRUE
    @grd4 pieces ≠ ∅
  then
    @act1 next :∈ pieces
    @act2 selection_step := 0
    @act3 selection_method := 0
    @act4 selection_inprogress := FALSE
end
```

Figure 3: Events corresponding to the criteria of the first selection in DAW.

```
event SP_SELECT_1
  where
    @grd1 ⊤
    @grd2 selection_method = 1
    @grd3 selection_step = 0
    @grd4 selection_inprogress = TRUE
  then
    @act1 selection_step := 1
end

event SP_SELECT_1_NEG
  where
    @grd1 ¬(⊤)
    @grd2 selection_method = 1
    @grd3 selection_step = 0
    @grd4 selection_inprogress = TRUE
  then
    @act1 selection_method := 2
    @act2 selection_step := 0
end
```

Figure 4: Events corresponding to the condition of the second selection in DAW, and its negation.

the second selection in DAW. As previously, the first guard of each event is the one that corresponds to the actual criteria, and here we note that **SP_SELECT_1_0** is quite complicated. In our translation we use a dictionary containing SPECTA statements used in conditions and criteria, and their equivalents in Event-B. This dictionary is reusable, but must still be updated as we add new translations.

We note that even though the two events named **SP_SELECT_0_COMPLETE** and **SP_SELECT_1_COMPLETE**, in Figures 3 and 5, respectively, appear to do the same thing, they could not be universally replaced by a single event. This is because in this particular case, both the first and second selection contain the same amount of criteria, which is not always the case.

```
event SP_SELECT_1_0
  any newpieces
  where
    @grd1 newpieces = { piece | piece ∈ pieces ∧
            (∀s · s ∈ pieces ∧ s ≠ piece ⇒
            (availability(s) ∗ (s − (current + buffersize)) ≥
            (availability(piece) ∗ (piece − (current + buffersize)))))}
    @grd2 selection_method = 1
    @grd3 selection_step = 1
    @grd4 selection_inprogress = TRUE
    @grd5 pieces ≠ ∅
  then
    @act1 pieces := newpieces
    @act2 selection_step := 2
end

event SP_SELECT_1_1
  any newpieces
  where
    @grd1 newpieces = { piece | piece ∈ pieces
            ∧ piece = min(pieces) }
    @grd2 selection_method = 1
    @grd3 selection_step = 2
    @grd4 selection_inprogress = TRUE
    @grd5 pieces ≠ ∅
  then
    @act1 pieces := newpieces
    @act2 selection_step := 3
end

event SP_SELECT_1_COMPLETE
  where
    @grd1 selection_method = 1
    @grd2 selection_step = 3
    @grd3 selection_inprogress = TRUE
    @grd4 pieces ≠ ∅
  then
    @act1 next :∈ pieces
    @act2 selection_step := 0
    @act3 selection_method := 0
    @act4 selection_inprogress := FALSE
end
```

Figure 5: Events corresponding to the criteria of the second selection in DAW.

```
event SP_SELECT_EMPTY
  where
    @grd1 selection_inprogress = TRUE
    @grd2 pieces = ∅
  then
    @act1 pieces := eligible
    @act2 selection_step := 0
    @act3 selection_method := selection_method + 1
end

event SP_SELECT_FAILED
  where
    @grd1 selection_inprogress = TRUE
    @grd2 selection_method = 2
  then
    @act1 pieces := eligible
    @act2 selection_step := 0
    @act3 selection_method := 0
    @act4 selection_inprogress := FALSE
end
```

Figure 6: Events corresponding to when no piece can be selected according to criteria and the whole algorithm is unable to select a piece.

We also get events corresponding to when no piece can be

selected according to criteria and when the whole algorithm is unable to select a piece, shown in Figure 6.

The events generated by translation can be put into an Event-B machine together with the variables, invariants specifying types of and relations between these, as well as an **INITIALISATION** event, as well as a context containing relevant constants and axioms about those. We can also add other events if needed, such as to update any variables modelling external conditions. Thus, from the translated code we can get a complete and correct Event-B model that can be expanded upon, for instance by adding the properties we would want to prove and which would cause us to choose a formal modelling environment for development. For both the BitTorrent and DAW algorithms we have created these models, and all the proof obligations generated were automatically discharged by the Rodin Platform tool.

3.3 Comparison

Event-B has previously been used for modelling a few different piece selection algorithms [26, 27]. Therefore, we can compare Event-B code from those to the code now generated from SPECTA using our simple translator. One substantial difference is that in previous work, a model of a distributed media streaming solution was built from the ground up, and early on an event was introduced for selecting the most prioritised piece. In case of more than one piece with the same priority, the piece with the lowest piece number is selected. This is common in several piece selection algorithms for streaming media, such as BiToS and DAW (4). How the priorities would actually be calculated is something that was specified only in the final refinement step, with separate events for different selection methods combined. For instance, in the case of DAW (4) there was one event setting the priority of pieces in the buffer and another event setting the priority of pieces outside the buffer. A side effect of this way of working with piece selection is that for an in-order selection algorithm priorities do not depend on any external information that could change, and thus each piece would always get the same priority in every iteration of the priority calculation. For modelling using translated SPECTA code, we did not introduce this calculation of priority for each piece. Instead, the way the piece selection should be carried out is something we left completely up to the translated SPECTA code.

For a more detailed comparison, we focused on DAW (4), which has been modelled in Event-B previously [26, 27]. For the sake of completeness, the models contain several additional events, modelling changing content availability, requesting and transferring pieces of content, as well as progress of content playback. Table 2 shows the number of generated proof obligations for the two models, and how many of these could be discharged automatically or interactively using version 3.0.1 of the Rodin Platform tool [14] and associated provers. Although both models contain a significant number of proof obligations, most of these were automatically discharged by the tool and therefore have no importance for the workload on the person modelling, although they can give us a hint towards the complexity of the model itself. The number of proof obligations that needed user intervention was low in both cases, although even lower in the case of the model based on translated SPECTA code. Additionally, in this model we needed fewer refinement steps in order to model the same functionality. However, the optimal number of refinement steps is highly subjective, and when comparing models there are also other factors that are hard to quantify objectively. For instance, we felt that the readability of the generated Event-B code was improved in the model based on translated SPECTA code compared to previous work. We found no differences in the overall behaviour of the two models when we used the ProB [19] Animator to animate the models.

In contrast to previous work on modelling piece selection, the events generated by translation from SPECTA are not optimised for any particular case. Thus, it would be possible to use SPECTA to generate Event-B code even for algorithms where calculating a numeric priority for each piece is not possible. Also, due the way we translate from SPECTA, it is possible to use content transfer algorithms with large numbers of criteria for each selection method. Finally, unlike previous work we could also here utilise algorithms other than ones explicitly made for transfer of streaming media content.

4. RELATED WORK

There is a lot of work in which different content transfer algorithms are described in informal and not standardised ways [11, 24, 25, 29]. Likewise, as mentioned in Section 1, pseudocode is not standardised even though there have been attempts at doing so [13]. There are also programming languages whose syntax is similar to pseudocode in that it is easily understood as a natural language, creating a situation where writing it requires learning a specific syntax but understanding existing code does not require specific knowledge about the syntax. One such language is AppleScript [4]. In general, we note that SPECTA is a domain-specific language, of which there are many others [16]. There has also been work on formally specifying a domain-specific pseudocode [9], which is more relevant to our problem. Another related field is programming by construction [21], in which programming concepts can be described in an independent manner and then later used to generate code in different programming languages. Finally, we note that in the content transfer algorithms that can be described using SPECTA the content must be split into pieces that can be individually requested, and fortunately this is already the case in many widely used approaches to content delivery. These include the previously mentioned BitTorrent [11] and related approaches, MPEG-Dynamic Adaptive Streaming (DASH) [28], as well as HTTP Live Streaming [18].

5. CONCLUSION

In this article we have described SPECTA, a language for formally specifying content transfer algorithms in an easily understandable manner, and shown that algorithms written in this language can be automatically translated into Event-B [2] code, as a starting point for analysis, verification, or further formal development. We have compared the formal model generated with converted SPECTA code to one developed separately in previous work, and noted that with a generic automatic translation comes increased flexibility, compared to manually generating similar code for a specific purpose.

Table 2: Proof obligations in our Event-B models.

Method	Total Proofs	Automatic	Interactive	Refinement Steps
Manual modelling of DAW [26]	94	92	2	5
DAW based on translation from SPECTA	76	75	1	2

In the future, we hope that the SPECTA language can be extended to support more keywords and operations. This includes information about other nodes, such as latency, transfer speed, and uptime. Such an extension would not only enable more advanced methods of selecting what piece of content to transfer, but also make it possible to specify which nodes should be involved in the transfer. As our SPECTA to Event-B translator is still fairly simple, we intend to rework it into a full-fledged plugin for the Rodin Platform tool [14], in order to make it more easy to integrate its use into the formal modelling workflow.

6. REFERENCES

[1] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[2] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[3] J.-R. Abrial, M. Butler, S. Hallerstede, T.-S. Hoang, F. Mehta, and L. Voisin. Rodin: An Open Toolset for Modelling and Reasoning in Event-B. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(6):447–466, 2010.

[4] AppleScript Language Guide. `https://developer.apple.com/library/mac/documentation/applescript/conceptual/applescriptlangguide/AppleScriptLanguageGuide.pdf` (Accessed January 2015).

[5] R. Back and R. Kurki-Suonio. Decentralization of Process Nets with Centralized Control. In *Proceedings of the 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 131–142, 1983.

[6] R. Back and K. Sere. Stepwise Refinement of Action Systems. *Structured Programming*, 12(1):17–30, 1991.

[7] R. Back and K. Sere. From Action Systems to Modular Systems. *Software - Concepts and Tools*, 17:26–39, 1996.

[8] R. Back and K. Sere. Superposition Refinement of Reactive Systems. *Formal Asp. Comput.*, 8(3):324–346, 1996.

[9] M. Backes, M. Berg, and D. Unruh. A Formal Language for Cryptographic Pseudocode. In I. Cervesato, H. Veith, and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 5330 of *Lecture Notes in Computer Science*, pages 353–376. Springer Berlin Heidelberg, 2008.

[10] K. P. Birman. *Guide to Reliable Distributed Systems: Building High-Assurance Applications and Cloud-Hosted Services*. Springer-Verlag London Limited, 2012.

[11] B. Cohen. Incentives Build Robustness in BitTorrent. In *1st Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[12] D. Craigen, S. Gerhart, and T. Ralson. Case Study: Paris Metro Signaling System. In *Proceedings of IEEE Software*, pages 32–35. IEEE, 1994.

[13] J. Dalbey. Pseudocode Standard. `http://users.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html` (Accessed January 2015), 2003.

[14] Event-B and the Rodin Platform. `http://www.event-b.org/` (Accessed January 2015).

[15] A. Fürst, T. S. Hoang, D. Basin, K. Desai, N. Sato, and K. Miyazaki. Code Generation for Event-B. In *The 11th International Conference on Integrated Formal Methods (iFM 2014)*, 2014.

[16] P. Hudak. Domain Specific Languages. In P. H. Salas, editor, *Handbook of Programming Languages, Vol. III: Little Languages and Tools*, chapter 3, pages 39–60. MacMillan, Indianapolis, 1998.

[17] S. Katz. A Superimposition Control Construct for Distributed Systems. *ACM Transactions on Programming Languages and Systems*, 15(2):337–356, April 1993.

[18] R. Pantos. HTTP Live Streaming. `http://tools.ietf.org/html/draft-pantos-http-live-streaming-13` (Accessed January 2015), 2014.

[19] The ProB Animator and Model Checker. `http://www.stups.uni-duesseldorf.de/ProB/` (Accessed January 2015).

[20] V. Rivera and N. Catano. The EventB2Java Rodin Plug-in. `http://poporo.uma.pt/EventB2Java/EventB2Java.html` (Accessed January 2015).

[21] M. Rönkkö, M. Stocker, M. Neovius, M. Kolehmainen, and L. Petre. Programming by Construction. Technical Report 1092, TUCS – Turku Centre for Computer Science, 2013. `http://tucs.fi/publications/view/?id=tRxStNeKoPe13a` (Accessed January 2015).

[22] P. Sandvik. SPECTA: A Formal Specification Language for Content Transfer Algorithms. In U. Wolter and Y. Lamo, editors, *24th Nordic Workshop on Programming Theory*, volume 403 of *Reports in Informatics*, pages 81–83. University of Bergen, 2012.

[23] P. Sandvik. SPECTA: A Formal Specification Language for Content Transfer Algorithms. In *IEEE Fifteenth International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2014)*. IEEE, 2014.

[24] P. Sandvik and M. Neovius. The Distance-Availability Weighted Piece Selection Method for BitTorrent: A BitTorrent Piece Selection Method for On-Demand Streaming. In A. Liotta, N. Antonopoulos, G. Exarchakos, and T. Hara, editors, *Proceedings of The First International Conference on Advances in*

P2P Systems (AP2PS 2009), pages 198–202. IEEE Computer Society, October 2009.

[25] P. Sandvik and M. Neovius. A Further Look at the Distance-Availability Weighted Piece Selection Method: A BitTorrent Piece Selection Method for On-Demand Media Streaming. *International Journal on Advances in Networks and Services*, 3(3 & 4):473–483, 2010.

[26] P. Sandvik and K. Sere. Formal Analysis and Verification of Peer-to-Peer Node Behaviour. In A. Liotta, N. Antonopoulos, G. Di Fatta, T. Hara, and Q. H. Vu, editors, *The Third International Conference on Advances in P2P Systems (AP2PS 2011)*, pages 47–52. IARIA, November 2011.

[27] P. Sandvik, K. Sere, and M. Waldén. An Event-B Model for On-Demand Streaming. Technical Report 994, TUCS – Turku Centre for Computer Science, December 2010. `http://tucs.fi/publications/view/?id=tSaSeWa10a` (Accessed January 2015).

[28] I. Sodagar. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE Multimedia*, 18:62–67, 2011.

[29] A. Vlavianos, M. Iliofotou, and M. Faloutsos. BiToS: Enhancing BitTorrent for Supporting Streaming Applications. In *9th IEEE Global Internet Symposium 2006*, April 2006.

[30] M. Waldén and K. Sere. Reasoning About Action Systems Using the B-Method. *Formal Methods in Systems Design*, 13:5–35, 1998.

[31] S. Wright. Automatic Generation of C from Event-B. In *Workshop on Integration of Model-based Formal Methods and Tools*, February 2009.

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
163. **Zhiyuan Yao**, Visual Customer Segmentation and Behavior Analysis – A SOM-Based Approach
164. **Timo Jolivet**, Combinatorics of Pisot Substitutions
165. **Rajeev Kumar Kanth**, Analysis and Life Cycle Assessment of Printed Antennas for Sustainable Wireless Systems
166. **Khalid Latif**, Design Space Exploration for MPSoC Architectures

167. **Bo Yang**, Towards Optimal Application Mapping for Energy-Efficient Many-Core Platforms
168. **Ali Hanzala Khan**, Consistency of UML Based Designs Using Ontology Reasoners
169. **Sonja Leskinen**, m-Equine: IS Support for the Horse Industry
170. **Fareed Ahmed Jokhio**, Video Transcoding in a Distributed Cloud Computing Environment
171. **Moazzam Fareed Niazi**, A Model-Based Development and Verification Framework for Distributed System-on-Chip Architecture
172. **Mari Huova**, Combinatorics on Words: New Aspects on Avoidability, Defect Effect, Equations and Palindromes
173. **Ville Timonen**, Scalable Algorithms for Height Field Illumination
174. **Henri Korvela**, Virtual Communities – A Virtual Treasure Trove for End-User Developers
175. **Kameswar Rao Vaddina**, Thermal-Aware Networked Many-Core Systems
176. **Janne Lahtiranta**, New and Emerging Challenges of the ICT-Mediated Health and Well-Being Services
177. **Irum Rauf**, Design and Validation of Stateful Composite RESTful Web Services
178. **Jari Björne**, Biomedical Event Extraction with Machine Learning
179. **Katri Haverinen**, Natural Language Processing Resources for Finnish: Corpus Development in the General and Clinical Domains
180. **Ville Salo**, Subshifts with Simple Cellular Automata
181. **Johan Ersfolk**, Scheduling Dynamic Dataflow Graphs
182. **Hongyan Liu**, On Advancing Business Intelligence in the Electricity Retail Market
183. **Adnan Ashraf**, Cost-Efficient Virtual Machine Management: Provisioning, Admission Control, and Consolidation
184. **Muhammad Nazrul Islam**, Design and Evaluation of Web Interface Signs to Improve Web Usability: A Semiotic Framework
185. **Johannes Tuikkala**, Algorithmic Techniques in Gene Expression Processing: From Imputation to Visualization
186. **Natalia Díaz Rodríguez**, Semantic and Fuzzy Modelling for Human Behaviour Recognition in Smart Spaces. A Case Study on Ambient Assisted Living
187. **Mikko Pänkäälä**, Potential and Challenges of Analog Reconfigurable Computation in Modern and Future CMOS
188. **Sami Hyrynsalmi**, Letters from the War of Ecosystems – An Analysis of Independent Software Vendors in Mobile Application Marketplaces
189. **Seppo Pulkkinen**, Efficient Optimization Algorithms for Nonlinear Data Analysis
190. **Sami Pyöttiälä**, Optimization and Measuring Techniques for Collect-and-Place Machines in Printed Circuit Board Industry
191. **Syed Mohammad Asad Hassan Jafri**, Virtual Runtime Application Partitions for Resource Management in Massively Parallel Architectures
192. **Toni Ernvall**, On Distributed Storage Codes
193. **Yuliya Prokhorova**, Rigorous Development of Safety-Critical Systems
194. **Olli Lahdenoja**, Local Binary Patterns in Focal-Plane Processing – Analysis and Applications
195. **Annika H. Holmbom**, Visual Analytics for Behavioral and Niche Market Segmentation
196. **Sergey Ostroumov**, Agent-Based Management System for Many-Core Platforms: Rigorous Design and Efficient Implementation
197. **Espen Suenson**, How Computer Programmers Work – Understanding Software Development in Practise
198. **Tuomas Poikela**, Readout Architectures for Hybrid Pixel Detector Readout Chips
199. **Bogdan Iancu**, Quantitative Refinement of Reaction-Based Biomodels
200. **Ilkka Törmä**, Structural and Computational Existence Results for Multidimensional Subshifts
201. **Sebastian Okser**, Scalable Feature Selection Applications for Genome-Wide Association Studies of Complex Diseases
202. **Fredrik Abbors**, Model-Based Testing of Software Systems: Functionality and Performance
203. **Inna Pereverzeva**, Formal Development of Resilient Distributed Systems
204. **Mikhail Barash**, Defining Contexts in Context-Free Grammars
205. **Sepinoud Azimi**, Computational Models for and from Biology: Simple Gene Assembly and Reaction Systems
206. **Petter Sandvik**, Formal Modelling for Digital Media Distribution

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
Faculty of Science and Engineering
- Computer Engineering
- Computer Science

Faculty of Social Sciences, Business and Economics
- Information Systems

ISBN 978-952-12-3294-7
ISSN 1239-1883

Petter Sandvik

Formal Modelling for Digital Media Distribution

Petter Sandvik

Formal Modelling for Digital Media Distribution

Petter Sandvik

Formal Modelling for Digital Media Distribution