An Architecture for Task Execution in Adverse Environments

Filip MILETIĆ

An Architecture for Task Execution in Adverse Environments

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. dr. ir. J. T. Fokkema,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op maandag 4 juni 2007 om 12.30 uur,

door Filip MILETIĆ

Electrical Engineer van de Universiteit van Belgrado, Servië
geboren te Kruševac, Servië.

Dit proefschrift is goedgekeurd door de promotor:

Prof. dr. ir. P. M. Dewilde

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof. dr. ir. P. M. Dewilde	Technische Universiteit Delft, promotor
Prof. dr. M. Prokin	Universiteit van Belgrado
Prof. dr. ir. A. J. van der Veen	Technische Universiteit Delft
Prof. dr. ir. F. C. A. Groen	Universiteit van Amsterdam
Prof. dr. ir. I. G. M. M. Niemegeers	Technische Universiteit Delft
dr. drs. L. J. M. Rothkrantz	Technische Universiteit Delft
dr. K. Nieuwenhuis	DECIS
Prof. dr. K. G. W. Goossens	Technische Universiteit Delft, reservelid

ISBN: 978-90-9021920-2

To Milan

Contents

Chapter 1

Introduction

This thesis describes an architecture for distributed computation in mobile environments. Here, the *workflow* stands for the operational aspect of a work procedure. This notion subsumes the structure of the tasks, who performs them, what their operation structure is, how they are synchronized, how information flows to support the tasks and how they are tracked. The designed architecture is tested in a proof-of-concept implementation named Distributed Workflow Execution Architecture for Mobile (DWEAM).

The interest in distributed workflow [90] execution architectures is long standing (some examples thereof are given further in the text). However, there has not yet been an adequate treatment of workflow execution in mobile systems. The benefits of such a system would be collected by users who perform a coordinated task in a complex environment and must establish own coordination infrastructure to do so. Typical users are the members of emergency rescue teams, i.e. police, fire brigade or medical personnel, when handling an incident, where the cooperation between the team members is hindered by mobility and adverse communication conditions. The communication via the Global System for Mobile (GSM) networks, that are used in small scale operations in urban areas cannot offer appropriate quality of service in face of escalation or infrastructure damage. The now-aged broadcast radio (i.e. walkie-talkie) typically does not support the coupling with information systems, as it is intended for the communication between humans. We therefore consider a system architecture that addresses the communication issues, while being infrastructure-independent and supporting both the work procedures for human operators, as well as that of the information systems.

In Europe, the market penetration percentage for GSM devices has long surpassed 80%, and tending to 100% [6], and the market drive thus created motivates the producers to equip their products with ever more computing power. This resulted in the advent of Personal Digital Assistants (PDA) and the convergence of the two technologies in the near future seems imminent. Assuming that in the near future mobile computing devices with communication facilities would be

routinely used, we recognize the potential for improving cooperative work that these devices offer. We consider this enough motivation to investigate and design a system that takes advantage of such devices. The example use case involving assistance to emergency services is just one of many possible future uses.

1.1 Outline of This Chapter

This introductory chapter begins the thesis by introducing the background of the problem in Section 1.2. The Section 1.4 states the research problem. Following a brief discussion of the preliminaries, we formulate the research question, and give its decomposition to three sub-problems. The suggested solutions to the research sub-problems are stated thereafter, as well as the improvements to the state of the art as given by this thesis. Finally, the Section 1.6 describes in brief the other chapters in this thesis.

1.2 Background

Distributed computing subsumes the topic of concurrent execution of algorithms on a collection of computing resources (processors, computers, nodes). We will be considering distributed computer systems in detail, so abbreviations "computer system" and "system" will always stand for such distributed systems.

Distributed computing was initially intended for large computation tasks associated with research in natural sciences, mathematics and like areas. Since the advent of the world-wide Internet, forms of distributed computing have become parts of daily lives. Trade, commerce, banking, mailing, along with a line of other ages old activities, were adorned with the prefix "*e*-" to reflect their new alliance with the information technology. The Internet has thus brought distributed computing to the desktop. The wide spread use of computers and the global interconnection such as the Internet gave a natural motivation to harness the large collective computing power. Example projects are Distributed Net (brute force code breaking, see [2]), Seti@Home (a search for artificial signals coming from deep space [74]), and Folding@Home (protein folding [31]). The University of California at Berkeley maintains BOINC [13], a framework to support this kind of "bona-fide collective" distributed computing. These examples show how computers worldwide can be joined together to perform a large task.

A promising new field for distributed computing is formed with the advent of mobile computers. GSM phones and PDAs have entered the scene during the nineties, likely to be followed by other smart wearable devices. They populated the earth in even greater numbers than personal computers, and it is expected that the growth would continue in the upcoming years. Apart from their initial applications, these devices are usually fully capable machines, every more often able to communicate with like machines in their vicinity. It can be inferred that these machines, when properly joined together, could perform significant

computational tasks for the collective of their owners and offer a range of new applications. For this vision to become a reality, a host of issues must be resolved so that *effective* computation becomes possible.

An informal notion of an effective computation is that which takes on a suitably posed question and produces an answer obtained by a sequence of well-defined primitive steps (i.e. an *algorithm*) to yield an answer with some predefined quality properties; additionally, presentation qualities may be involved, as specified by the Human-Computer Interface (HCI) guidelines. The quality properties define the manner in which the computer system produces an answer: how much time it takes whether it is of a good enough quality etc. As Gärtner [33] reported, in 1977 Lamport [48] observed that the system properties can be be put into two distinct classes: safety properties and liveness properties. The safety properties state that "something bad never happens", i.e. that under no condition the system is to enter an unwanted state. The liveness properties state that "eventually something good happens", i.e. that the system eventually reaches some state with favorable properties (e.g. a state in which the computational outcome is presented). Effectiveness is therefore restated in terms of both liveness and safety, as well as subjective measurement of the usefulness of the system, as perceived by the users. By liveness, an effective system must present the user with an answer (if the result can be computed given the system's capabilities). By safety, the result must be correct, supplied within a given time frame, incurring a limited resource cost, possibly others. By HCI requirements the system must facilitate HCI by timely presentation, detail abstraction and ease of use.

A computer system's effectiveness fundamentally depends on its architecture. It is thus important to have at hand a categorization of distributed architectures. Being able to classify our system informs us of its possibilities. A choice of the architecture will determine the set of computational tasks that the computer system can and can not perform. Allowing the possibility of faults in the computation further segregates the possible from the impossible. According to Lynch ([51], Chapter 1), the distributed algorithms can differ by a number of attributes ultimately determined by the architectural choices:

1. *The communication method.* This concerns distributed algorithms running on a collection of processors that must communicate somehow. Common methods of communication include addressing shared memory, sending point-to-point or broadcast messages and executing remote procedure calls. The ordering of messages can be important too.

2. *The timing model.* This concerns the manner in which different processors execute their separate tasks. At one extreme, the processors run in lockstep, progressing in perfect synchrony. At the other extreme, they can each run at various relative speeds and can take arbitrary execution turns. In between the extremes are various partially synchronous systems, in which the processors have partial information about timing.

3. *The failure model.* This concerns the way that the hardware may fail to perform its tasks. It can be assumed to be completely reliable, yielding a fault-free assumption. Or it may be required to tolerate some fraction of failures.

4. *The addressed problems.* This concerns the problems that one attempts to solve by a distributed system. Typical problems of this sort are *resource allocation*, *concurrency control*, *deadlock detection*, *global snapshots*, *synchronization* and *implementation of various types of objects.*

The properties are implied by the adopted use case. The communication method employed in our approach (and the DWEAM system) is *localized point-to-point* as the nodes are assumed to communicate only with other geographically close nodes. The timing model is *virtually synchronous* as the nodes run independently except for few points where message exchange takes place. The failure model we consider is *communication and node fail-stop*, as the nodes and their interconnections can fail (and recover) at runtime. Finally the addressed problems concern the *efficiency* of the computation in such an environment.

A World of (Im)possibilities

The effects of system attributes with respect to the classification given by Lynch were succinctly commented in a paper by Turek and Shasha [43], by analyzing a fictitious storyline named "The Parable of La Tryste". They comment the solution of the prototypical *consensus problem* that arises in distributed computing. They explain when consensus problem is solvable, when it is not, and when unsolvable, how the problem can be relaxed so that a solution can be found.

> "Bob and Alice have discovered that they have a lot in common. For example, they both prefer e-mail to telephone. On a cold winter day, Alice sends Bob electronic mail at 10a.m. saying 'Let's meet at noon in front of La Tryste.'"
>
> "The e-mail connection between our two protagonists is known to lose messages, but today they are lucky and Alice's message arrives at Bob's workstation at 10:20a.m. Bob looks at his calendar and sees he is free for lunch. So he sends an acknowledgment."
>
> "Alice receives the acknowledgment at 10:45a.m. and prepares to go out, when a thought occurs to her: 'If Bob doesn't know that I received his acknowledgment, he might think I won't wait for him. I'd better acknowledge his acknowledgment.'"
>
> "And so it goes. We can show that, ultimately, neither Bob nor Alice will make it to La Tryste unless at least one of them is willing to risk waiting in the cold without meeting the other."

The "La Tryste" scenario demonstrates the difficulty of reaching a consensus in systems with asynchronous operation, unbounded message delays, and possible message loss. The problem is that as sending the message can fail, neither Bob nor Alice can know whether their decision is communicated to the other. In fact, under these conditions, Lynch et al. have proved that consensus is impossible [52].

This result brings us to an apparent conflict with what we perceive in daily life. Despite the impossibility of communication, people and computers do manage to cooperate and things do get done. The apparent conflict comes from the way the problem is analyzed. Given that no errors in communication are allowed, and the fact that no matter how many times the message exchange is repeated, we are never able to rule out the worst case (as all communication eventually ends in an error), regardless of how improbable it might be. A conclusion follows that the problem as posed has no solution. Thus the requirement on reliable communication must be relaxed for the result to be applicable in the real world. The outline of this approach is given in the following Section.

The Information Theoretic Approach

Now turn to the area of information theory, where a similar problem has long been solved by Shannon [75] (numerous others followed) who formulated the communication problem and gave a set of initial solutions. A Message chosen from a predetermined fixed alphabet is transferred from an Information Source to a Destination (see Figure 1.1). Before sending, the Message is encoded for transmission

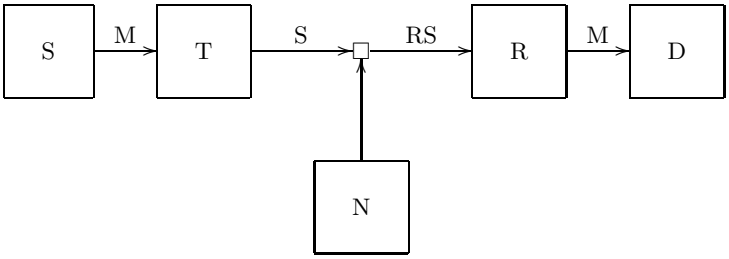


Figure 1.1: The communication system setup, as given by Shannon [75]. S: Source. T: Transmitter. R: Receiver, N: Noise, D: Destination. M: Message , S: Signal, RS: Received Signal.

by a Transmitter where it becomes a Signal. The Signal is sent through a communication channel, where it is inevitably affected by noise (its effect modeled by a Noise Source). The Receiver obtains a likely distorted Received Signal, and decodes it so as to obtain a Message which is finally forwarded to the destination.

The difference with the "La Tryste" is that the probabilities of communication failure are taken into account. Shannon's conclusion from here is that provided the communication channel is used the right way, the communication could be

made as reliable as needed. However, the price one pays for this is that in principle more information needs to be sent over the channel than is minimally required to describe the message.

Example 1 (Binary Symmetric Channel) *Let us temporarily diverge from the original "La Tryste" and consider the example of a* binary symmetric channel*, as given in Figure 1.2. In this example, the communication channel is binary, i.e.*

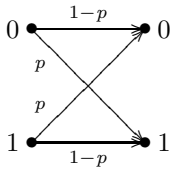


Figure 1.2: The binary symmetric channel model, with error probability p per channel use.

two distinct messages can be sent, and only one of the two can be sent at a time. Call these "1" and "0", or in the case of "La Tryste", call them "yes, let's meet", and "sorry, I'm busy". When the Information Source can produce more than two distinct messages, they can be sent by using the binary channel several times, so that different sequences consisting of zeros and ones denote different messages. The total of $\lceil \log_2 M \rceil$ *channel uses are needed so that there are at least* M *distinct binary sequences. This is a familiar scenario frequently occurring in digital communications, including that between computer systems.*

Due to noise, each channel use can be distorted, so that a "1" that has been sent becomes a "0" or vice-versa. This happens with some probability p*. Thus the probability is* $1 - p$ *that one channel use ends with no errors. This parallels the possibility that a "yes" might be flipped to "no" in transit, or other way around. Shannon determined that there is a fundamental limit for the rate (in bits per channel use) at which one can transmit information through this (or for that matter, any other) channel, which he called the* capacity*. For the binary symmetric channel it is given by [21]:*

$$C = H(p) = -p \log_2 p - (1-p) \log_2 (1-p).$$

In this expression, $H(p)$ *is the* entropy *of a binary source. Moreover, it was determined that transmitting through such a channel cannot be done at a rate higher than* C*. The capacity of a binary channel with respect to the probability* p *is given in Figure 1.3.*

What use is this for Alice and Bob? First one sees that in most of the cases (except at the endpoints of the graph in Figure 1.3 corresponding to $p = 0$ and $p = 1$), they cannot expect to transfer a whole message (i.e. "fully" agree whether

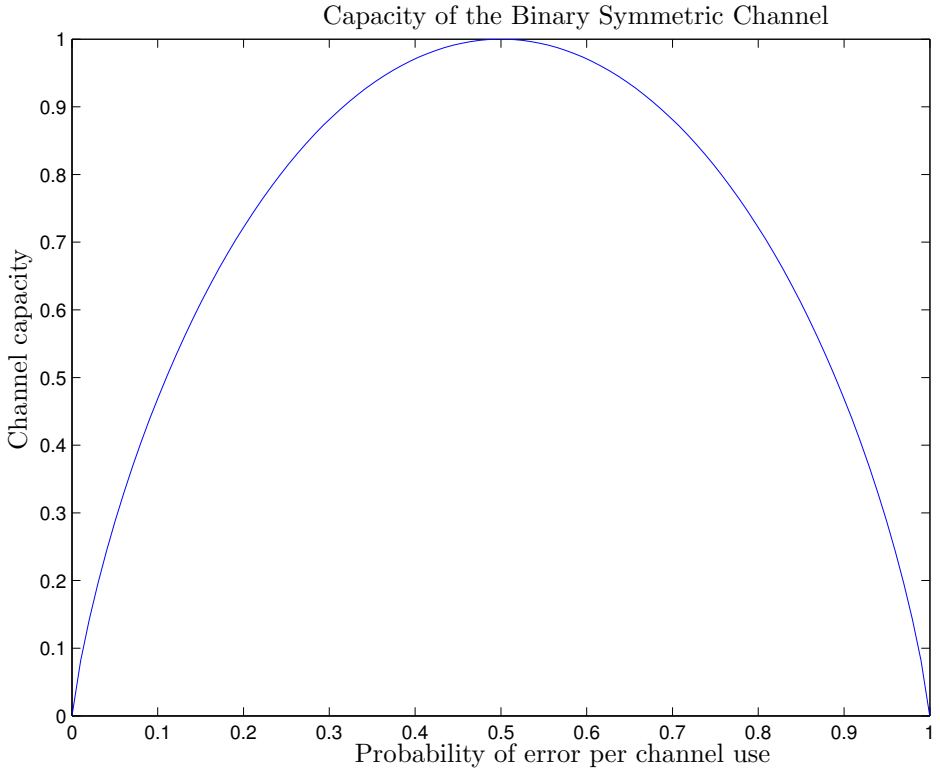


Figure 1.3: The capacity of the binary symmetric channel in bits per channel use, versus the probability p of error per channel use.

a meeting goes through or not) by just a single channel use. By multiple channel use they could repeat the agreement over and over and then take a majority vote. But then, their information rate (i.e. say the price they pay for the communication per unit transferred messages) will increase with the number of repetitions, which is also not good.

However, if Alice and Bob agree to set not one, but several meetings at once, it would be possible to agree on $C \cdot m$ independent meetings by exchanging only m messages (and therefore paying the amount proportional to m for the conversation), and do this with as small probability of error as desired. In reality, driving the error probability close to zero would also mean having to agree on an impossibly long sequence of meetings at once, but this realistic complication is not considered important in the model.

Example 2 (Binary Erasure Channel) *Consider now a setup that is somewhat different and closer to "La Tryste". The new channel is shown in Figure 1.4.*

For each sent binary symbol, there is a probability p that it gets lost in the chan-

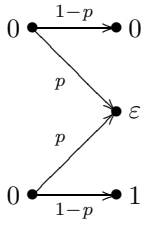


Figure 1.4: The binary erasure channel [75].

nel. The loss is denoted at the Receiver by a third symbol ε. Otherwise, with the probability $1 - p$ the symbol is received unchanged.

This channel model is better for computer networks than the binary symmetric channel. The computer networks are commonly packet-switched*, with messages of various origins being multiplexed and sent through the same channel, only for this process to be reversed at the other end. The packet-switches operate by stacking all the transmissions pending delivery in an outbound queue, which is emptied as packets are being sent. As queues have limited capacity, it is possible that a packet arriving at a switch with a full transmit queue will get dropped. Such omissions are be perceived as erasures at the receiver. In this case, the channel capacity is given by a simpler expression:*

$$C = 1 - p,$$

which is at the same time the expected fraction of lossless transmissions.

The contrast of "La Tryste" and the communication examples 1 and 2 shows important differences in approach. The impossibility result illustrated in the former employs a pessimistic analysis, whereby only the worst case is considered. On the other hand, the communication example takes into account a more realistic scenario and an advanced approach yielding provably better performance, at the cost of an (arbitrarily small) probability of error. One can note the use of the statistical properties of the communication channel to yield the desired performance.

The statistical properties of the environment will come back at several points in the analyses given in the upcoming chapters.

1.3 Properties

The Resources

Building a computation platform out of mobile devices connected by wireless network somewhat from the design of conventional (fixed) computer networks.

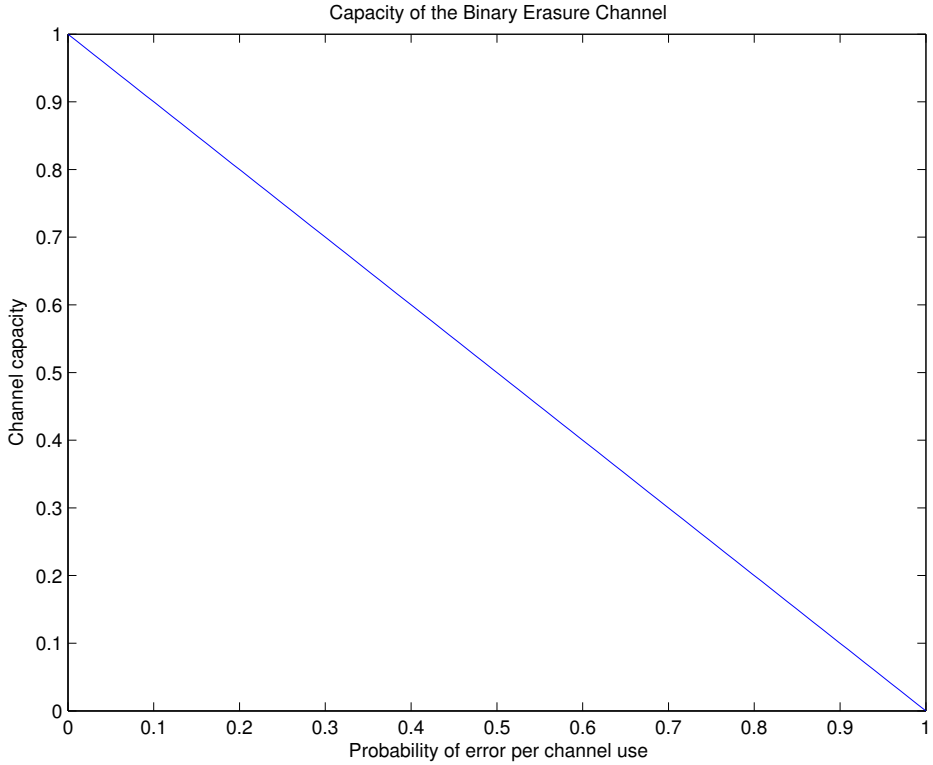


Figure 1.5: The capacity of the binary erasure channel in bits per channel use, versus the probability p of error per channel use.

The mobile computers are more "immersed" into the real world and are more susceptible to environment influences than their fixed counterparts. Taking into account the interaction with the environment forms therefore an important part of the mobile platform design.

A mobile computational platform consists of two types of resources:

1. *Computation resources (computers, nodes).* These resources apply a generic function on data. A function that a node can apply is said to be *implemented* by that node. A single node can implement several distinct functions at a given time.

2. *Communication resources (network).* The network connects computing resources, enabling them to send data objects to each other. Due to the properties of radio communication (the interference and path-loss) each node is only able to contact the immediate neighbours directly. Contacting a node that is further away is only achievable via intermediate nodes.

The special properties of the mobile computing platform, constituting the design constraints, are:

1. *Node volatility.* Mobile nodes are power-constrained. Thus they can run out of power and forcibly become unreachable. The nodes can also be destroyed. This constitutes a *involuntary leave*. Nodes can also be shut down, constituting a *voluntary leave*.

2. *Network volatility.* Due to path-loss and interference in the radio-based network, and the fact that the nodes change position over time, each individual link between pairs of nodes is subject to variation.

The Tasks

The mobile platform is assembled so that some predefined computational task can be performed. Interesting tasks for distributed execution are such that can be decomposed into loosely coupled subtasks. Thus, the patterns of cooperation between nodes are not random. Rather, they follow a specific pattern, as determined by the task dependency. Furthermore, an allocation schema must exist whereby tasks are allocated for execution (i.e. *mapped*) to nodes.

Thus the task execution is characterized by the following properties:

1. *Static structure.* This concerns the way the tasks relate to each other, i.e. how they are subordinated what their mutual dependencies are, and which operations on the data they perform.

2. *Dynamic structure.* This concerns the rules by which the tasks are executed, and the management of the data flow between dependent tasks.

3. *Mapping to nodes.* This concerns allocating tasks to nodes and handling the volatility.

These properties taken together constitute a *workflow*.

The Users

An important design goal for DWEAM is the easy integration of the users input in the overall system works. The user experience is not treated, as it is considered an HCI issue out of scope of this thesis. However, providing an uniform interface for the coupling of the machine generated and user-supplied result is important. To distinguish between these two classes of system participants we use the term *agent*, when a machine participant in the system is meant and *actor* for a human participant in the system. Together they are all named *workers*.

1.4 Problem Statement

We consider, in essence, a distributed computing system intended to execute a specified abstract computing *task*, under resource volatility. We represent the computing task by a workflow. The workers are capable of executing a subset of tasks in a workflow.

1. *Execution speed.* Multiple tasks are executed in parallel. This can not only be faster when compared to sequential processing for well-structured tasks, but can also scale with the number of available workers.

2. *Separation of concern.* It is possible to map special processes to workers with special capabilities. A failed agent can be replaced with one having comparable capabilities. A task intended for an absent actor can be re-assigned.

3. *Mixed initiative system support.* It is possible to mix the active participation of the agents and actors in task execution.

Research Question

The thesis is the answer to a single research question: *How is computation structured and controlled in this environment?*

The research question decomposes into interwoven sub-problems:

1. *The Environment Model.* An environment model is needed to express the environment influence to DWEAM performance.

   What is an appropriate model for the environment of this distributed system? What are its properties and what design patterns can be derived from it?

2. *The Storage Model.* The storage resources accessible to the entire system vary over time. Thus in addition to conventional data storage mechanisms, special care must be taken to ensure storage availability despite the time variations.

   What does this mechanism look like? What is its performance?

3. *The Execution Model.* The computational resources offered to the entire system vary over time. Thus in addition to conventional execution mechanisms, special care must be taken to ensure the computational resource availability despite time variations.

   What does this mechanism look like? What is its performance?

1.5 Contributions

The solution components answer in turn each of the three research sub-problems. They form the core contributions of this thesis.

1. *The Erasure Graph* answers the Environment Model question. It captures the properties of the environment the system nodes are embedded in. The analysis of the Erasure Graph yields performance bounds for both the Storage and Execution model.

2. *The Distributed Coding* answers the Storage Model question. In view of the Erasure Graph model, a technique is devised for resource-efficient and fault-tolerant data distribution. In this technique, we consider the serialization of data into binary streams, and the distribution of parts of these streams to independent nodes. We give the estimate of the storage capacity for the model thus obtained.

3. *The Event Notification* answers the Execution Model question. In view of the Erasure Graph model, a technique is devised for dynamic workflow assembly, maintenance and execution. The event notification is based on the Core Based Tree (CBT), a tree-like structure that ensures all the required connections can be established. The performance of the CBT is estimated The connections are established using *content-based addressing* whereby for communication, the contents of the messages are used to route them to all the intended recipients. Finally, the execution model is formulated that guarantees the execution of the CPN implemented by the nodes participating in the event notification structure.

The solution components are being implemented into DWEAM, the proof-of-concept system.

Advances with Respect to the State of the Art

The contributions to the state of the art, made in this thesis, are as follows.

1. The formulation of distributed computation in volatile environments as a distributed execution of a CPN, and its description in terms of the Object-Z language (see Chapter 2). This contribution enabled us to identify that token preservation and object delivery were needed to ensure the distributed workflow execution.

2. The formalism casting the informally described tasks with the foremost aim to enable the formalization of tasks performed by the emergency rescue teams from the context of Chaotic Open-World Multi-Agent Based Intelligent Networked Decision Support System (Combined) Systems into the CPN form (see Chapter 4). This contribution enabled us to leave the details of the application domain behind and concentrate to distributed CPN execution.

3. The *environment and storage models*, which enabled the discussion about the token preservation schema (see Chapter 5). We introduced the concept of *erasure graph* and computed the capacity

4. The *CBT construction algorithm*, which builds the basic interconnection structure for the solution of the distributed Service Discovery Problem (see Chapter 6).

5. The *execution model*, stemming from the solution of the *distributed Service Discovery Problem (SDP)*. Within it, the content-based *matching algorithm* is used to establish the object delivery rules in a wireless network (see Chapter 7).

All the contributions have found their way in the implementation of DWEAM. The implementation amounted to somewhat more than 60 thousand lines of code and documentation, written in Java, Scheme, XML and Javadoc. According to the *sloccount* utility [88], the development effort estimate of the implementation according to the *Basic COCOMO*[1] model [12] amounted to 7.4 person-years, and would cost about 1 million USD to develop in 1.15 years by an average of 6.46 developers[2]. The author's contributions to the Combined code base had been excluded from this estimation. This was done as the contributions there are mixed with those of other developers of the Combined code base. They are therefore difficult to tell apart, as versions of the same packages were written and updated by multiple authors.

Publications

The work on DWEAM has produced the following publications:

1. Filip Miletic and Patrick Dewilde. A distributed structure for service description forwarding in mobile multi-agent systems. *Intl. Tran. Systems Science and Applications*, 2(3):227–244, 2006

2. Filip Miletic and Patrick Dewilde. Design considerations for an infrastructureless mobile middleware platform. In Katja Verbeeck, Karl Tuyls, Ann Nowé, Bernard Manderick, and Bart Kuijpers, editors, *BNAIC*, pages 174–179. Koninklijke Vlaamse Academie van Belgie voor Wetenschappen en Kunsten, 2005

3. Filip Miletic and Patrick Dewilde. Data storage in unreliable multi-agent networks. In Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh, and Michael Wooldridge, editors, *AAMAS*, pages 1339–1340. ACM, 2005

[1] Constructive Cost Model, an estimation method for person-months needed for completing a software project.

[2] This figure is obtained as the quotient of the effort and the schedule, as obviously the number of developers in reality can only be an integer.

4. Filip Miletić and Patrick Dewilde. Coding approach to fault tolerance in multi-agent systems. In *IEEE Conference on Knowledge Intensive Multiagent Systems*. IEEE, April 2005

5. Filip Miletić and Patrick Dewilde. Distributed coding in multiagent systems. In *IEEE Conference on Systems, Man and Cybernetics*. IEEE, October 2004

1.6 Outline of The Thesis

In the thesis body, we first present the toolkit to be used later in the exposition. We then take on in sequence the components of the research sub-problem and treat them in depth.

Chapter 2

This Chapter contains the thesis preliminaries. First, the DWEAM problem is put into a broader context of mixed-initiative multi-actor, multi-agent systems. It is seen that DWEAM is but a single component of a larger system, called Combined. The unifying description of Combined is given, and the requirements for the integrated Combined system are given. Following this description, we give an overview and commentary of the related work. We describe the toolkit that is used in the subsequent chapters, which relies on the usage of the Object-Z language, and the process model of the CPNs. We then formalize the task of DWEAM, and complete the description of the used framework by specifying the distributed blackboard.

Chapter 3

This Chapter contains the overview of the architecture of DWEAM. In this Chapter, the basic notions used in the remainder of the thesis are introduced and explained in a nutshell. Here we also present the layered structure of DWEAM.

Chapter 4

This Chapter contains the method used to cast the informal descriptions of interrelated tasks into a CPN. The CPN description is then converted into an implementation using a distributed blackboard.

Chapter 5

This Chapter contains the description of the operating environment and the storage model. We investigate the connectivity function for a set of nodes in two dimensions. Having done that, we turn our attention to the storage model, where we estimate the performance of the data partitioning to achieve the token preservation.

Chapter 6

This Chapter contains the detailed description of the CBT construction algorithm. The notions of *producers* and *consumers* are introduced, and the SDP is defined. The CBT is afterwards used to provide a solution to the SDP, by finding the producers and consumers which are *compatible*, i.e. those that communicate through the delivery of data objects. A detailed analysis of the CBT construction algorithm is given, with the CPN descriptions of its phases and the discussion about its performance.

Chapter 7

This Chapter describes the *Dataspace* model to which all the data distributed in DWEAM must conform. Thereafter, the *matching algorithm* is given. In it, the CBT structure from Chapter 6 is used to compute the matching between the compatible producers and consumers. The proof of the matching algorithm is given, followed by the CPN description of the implementation.

Chapter 8

This Chapter lists the contributions of the thesis, explains the outlook of distributed workflow execution in the contemporary context, and gives pointers to future work.

Chapter 2

Toolkit

In this Chapter we present the mathematical toolkit used for formal specification throughout the thesis. The toolkit contains two inter-related tools. These are: *Object-Z*, a formal specification and documentation language which is used to describe system states and transitions; and *CPN*, a graphical formal language which is used to describe the concurrency which arises in DWEAM.

We dedicate a separate Chapter to the toolkit description in order to lay the foundation for the used notation, as well as to explain the toolkit's relations to other equivalent ways of modeling software components. Additionally, we motivate the reasons for the choice of the two used tools.

2.1 Introduction

Early in the development cycle we experienced the need for having a formal language to describe DWEAM. There are several levels at which the description needs to be available, each of those coupled with the intended use of the particular description, and the intended target audience. The descriptions differ among themselves in the form and the level of detail that they encompass, while at all times they are required to correspond to each other in those description components which are shared.

A number of intended uses can be identified for the system descriptions. These intended uses are very similar to those found elsewhere in software products.

1. *Execution.* This use subsumes the descriptions needed for the system to be represented inside in a way that can be directly executed by a computer. The description is given in the form of binary files, not intended to be read by humans. The executable form encodes full detail about system operation. It is therefore difficult and time consuming to recover other representation forms from this one.

2. *Development.* This use subsumes the descriptions used to expand the functionality of the system. The description is given in the form of source code files, which use a high level language to describe program functionality. With some exceptions, these files are written by programmers, but they are intended to be easily compilable into the executable form. These files can be readable for a human, although the pace at which the description can be understood depends greatly on the way the files are organized.

3. *Maintenance.* This use subsumes the descriptions needed so that the system can be repaired and upgraded. In order to clarify critical points in the development description, the maintenance notes are given in form of comments to the source code representation. The comments are used to clarify portions of program code, and in an increasing number of cases, also for the automatic documentation generation.

4. *Design.* This use subsumes the descriptions used to invent new functionality and reason about the properties of the system with the new functionality included. The form of the design can vary in format, level of detail and formality, and is the origin from which all the other descriptions are generated.

5. *Presentation.* This use subsumes the descriptions used to present in a condensed manner either the system composition, or the results of the system activities. The presentation is intended for human audience, and uses text, diagrams and formulas to emphasize the key design, or evaluation points. The level of detail is often adjusted to present only the relevant data for the presentation context.

Inspecting this list we can conclude that the descriptions pertaining to system design are the first documents that are produced about a system. They are the documents from which all other representations are produced, regardless of the level of detail employed and the intended use. It is therefore important that the design decisions be documented in an unambiguous way, lending themselves both to understanding and implementation. Formal specifications fulfill this goal well.

2.2 Description Quality Requirements

The formal methods invented for describing software systems vary in the intended purpose, the target audience and expressive power. The choice of the right formal representation is constrained by the need of it being able to fulfill the imposed requirements on the description quality. In the case of the DWEAM design, the description quality requirements are identified as follows:

1. *Expression economy.* The description must have a developed vocabulary supporting programmatic structures that often occur in software system design. This is to prevent having to define these familiar structures to

complete the specification. For further economy, the description needs to provide ways to reuse the description components, to ease the description understanding and the maintainability.

2. *Implementation neutrality.* The description must be detached from the implementation form of the description. This requirement is in line with the previous one, as it stipulates that the adopted description may use familiar idiomatic constructions regardless of whether they are idiomatic in the implementation representation.

3. *State representation.* The description must represent system states in a manageable way. It must allow partial state descriptions.

4. *Concurrency representation.* The description must allow explicit description of concurrency. The DWEAM is a system that critically depends on concurrent execution. This dependence must therefore be explicitly described.

5. *Executability.* There must be a straightforward way to transform the system description into an executable form. This must either be automatic, or manual, provided that the right procedure is followed.

6. *Openness to analysis.* The description must admit formal analyses and proof methods.

2.3 Representation with Object-Z and CPN

The representation that we adopted for specifying DWEAM in this thesis is a combination of two formal specification languages. The languages are Object-Z and Coloured Petri Net (CPN) were chosen for their merits with respect to the description quality requirements given in Section 2.2. According to [14] (Section 1.2.1), "Z is a typed language based on set theory and first order predicate logic", with Object-Z being an extension thereof incorporating language facilities lending themselves to the specification in the object-oriented style. The CPN [69] is a graphical representation language which is especially suitable for the description of concurrent execution of distributed algorithms. These two representations complement each other well for the description of concurrent systems.

Strictly speaking, in this thesis we use Object-Z as the basis for the formal specification, while the CPN is used as syntactic sugar to represent concurrency. This approach is due to Z's (hence also Object-Z's) lack of methods for explicit concurrency representation. The concurrency is hence handled by the constructs readily available within CPN, and the connection between the two representations is established by specifying the CPN semantics in Object-Z itself.

The account of the combined Object-Z and CPN description with respect to the representation quality requirements from Section 2.2 is given below.

1. *Familiarity.* Both Object-Z and CPN use notation that is well known. Object-Z uses the notation drawing from basic set theory that is common and well understood. Similarly, the CPN notation uses annotated graphs, another familiar device.

2. *Expression economy.* Although Object-Z's set-theoretic notation is basic, it also has a standard toolkit which supports structures more elaborate than those of the basic sets. To name a few: relations, functions, sequences and bags. Further, its object orientation allows the descriptions to be reused efficiently.

3. *Implementation neutrality.* Object-Z and CPN representations are not coupled to a particular implementation language. This is in contrast to modeling languages such as Unified Modeling Language (UML), in which descriptions of program semantics must be given in the implementation language (e.g. Java) as the modeling language itself cannot express it.

4. *State representation.* In Object-Z, the *schema notation* can be used to specify partial sets of system state variables, as well as state transitions. Facilities exist to denote the schema composition.

5. *Concurrency Representation.* The representation of concurrency is handled by CPN through explicit representation of the control flow by places and transitions.

6. *Executability.* The CPN specifications give precise instructions on the control flow of a distributed program, and the Object-Z description supplies the description of the data transformation at each of the CPN's transitions. The full specification can be implemented in a straightforward manner on a blackboard-based computer system using a set of simple compilation rules.

There exist similar and more complete takes on the specification of distributed systems, as given in [77], for instance, where the concurrency and process control has been handled by extending Z by Communicating Sequential Processes (CSP) [40], the process-oriented language invented by C. A. R. Hoare. As the integration of the state-oriented and process-oriented languages is an elaborate topic out of scope of this thesis, we provided the support for concurrency only to the extent required for the description of DWEAM.

The choice of the Object-Z and CPN combination that we opted for as the language of choice for the formal description of DWEAM is by no means unique. Equivalent and related approaches are numerous as is shown here.

2.4 Object-Z Description

The notation is based on the Z language (see [78]) and its object-oriented extension Object-Z. Z is a "typed formal specification language based on first order predicate

logic and Zermelo-Frankel (ZF) set theory" (from [14]) extended with a useful mathematical toolkit for expressing frequent constructs in computer science.

The Z notation includes the familiar symbols for predicate logic ($=, \neq$, *true*, *false*, $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\forall$, $\exists$, etc.) with widely understood meaning. Same goes for the sets and expressions ($\in, \notin, \cup, \cap, \subseteq$, etc.). The set of natural numbers is commonly denoted as $\mathbb{N}$, and the set of whole numbers is $\mathbb{Z}$. Set comprehension is denoted as: $\{\ x \mid P(x)\ \}$, and reads as "set of elements x with the property $P(x)$". Element comprehension is denoted as: $\mu\, x \bullet P(x)$ and reads as: "The unique element x that solves the equation $x = P(x)$". Substitution is supported by the Lambda-notation. A function object that increases a given number by one reads: $\lambda\, x : \mathbb{Z} \bullet x + 1$. Distinct identifiers are denoted by different strings. Examples of legal identifier names are: $a, b, c, A, B, a_1, b_1, \alpha, \beta, word, Car_1, \ldots$.

Every variable in Z has a type, i.e. a set from which it is drawn, and which must match when associated with other variables. For a variable a of some type M, one writes: $a : M$. An *opaque type* can be introduced, i.e. such that its properties are abstracted at introduction point. This is written $[\ M\ ]$ and allows the use of M as a type identifier in subsequent text. We reserve the initial-capital words for the type names, e.g. *Task*.

When a variable is a set itself, its type is the set of sets. For S a type, the set of subsets of S is written as $\mathbb{P}\, S$.

A relation $\underline{R}$ between two sets P and Q is a subset of $P \times Q$. This can take an infix form $p \underline{R} q$, for $(p, q) \in \underline{R}$, provided $\underline{R}$ is a relation between the elements of P and Q. It is defined by an axiomatic schema:

$$\begin{array}{|l} \_\underline{R}\_ : P \leftrightarrow Q \\ \hline W \end{array}$$

with W a predicate on $\underline{R}$. The domain of a relation $\underline{R}$: $T \leftrightarrow U$ (written: dom $\underline{R}$) is the set of elements of T that are related to at least one element in U. The range of a relation $\underline{R}$ (written: ran $\underline{R}$) is the set of elements of U that are related to at least one element of $\underline{R}$. The underscores (_)in the above specification are argument placeholders: given $\_\underline{R}\_$, when $p \underline{R} q$ is used to claim that p and q are in the relation $\underline{R}$, the types of p and q are implicitly taken to be the types appearing in the type definition of the relation $\underline{R}$. Hence the type of p must be P and the type of q must be Q. The inverse of a relation is denoted as $\underline{R}^{\sim}$ and it is obtained by reversing all the tuples from $\underline{R}$. A transitive closure of the relation $\underline{R}$ (i.e. $\underline{R} \cup \underline{R}^2 \cup \ldots$) is denoted by $\underline{R}^+$. For a set A from dom $\underline{R}$, $A \lhd \underline{R}$ is the *domain restriction*, i.e. the subset of elements of $\underline{R}$ whose first components are elements of A. Similarly for B from ran $\underline{R}$, the expression $\underline{R} \rhd B$ gives the *range restriction*, with analogous meaning. Domain and range *anti-restriction* are denoted by $\_ \lhd\!\!\!\!- \_$ and $\_ \rhd\!\!\!\!- \_$ respectively. $\underline{R}\ (\!|\ A\ |\!)$ is the *relational image* of $\underline{R}$ with respect to the set A.

A function is a special form of a relation in which each element in the domain has at most one element associated with it. A function F from a set T to U is

defined by a modified axiomatic schema:

$$\mid F : T \to U$$

Partial functions F, where some dom T do not have an image in U are given as $F : T \pfun U$. A pair $(t : T, u : U)$ from F can be expressed in a more graphical way as $t \mapsto u$, the *maplet notation*.

A shorthand is introduced by using the == connective. Thus $V == \{u, v\}$ makes V a shorthand for a two element set $\{u, v\}$. A theorem is denoted as:

$$\Gamma \vdash P$$

where Γ is the context (or none, if the context is global), and P a property that is proven. It is read: "in Γ, P is valid."

The forward composition of two functions F and G is denoted as $F \fatsemi G$.

$$\vdash \operatorname{ran} F = \operatorname{dom} G \Rightarrow F \fatsemi G = \{y \mid \forall x \in \operatorname{dom} F \bullet y = GFx\}$$

A bijection G is denoted as:

$$\mid G : T \rightarrowtail U$$

The *schema* notation is used to structure specifications. The example below gives the schema *Book* (from [14], Section 3.6) . The opaque types *People* and *CHAR* must be defined for completeness, as they are used in the schema.

$$[People, CHAR]$$

Book

$$\begin{array}{l} author : People \\ title : \operatorname{seq} CHAR \\ readership : \mathbb{P}\, People \\ rating : People \pfun 0 \mathinner{..} 10 \\ \hline readership = \operatorname{dom} rating \end{array}$$

A schema can also be written in line, so the following is the same as above:

$$Book \mathrel{\widehat=} [\ author : People;\ title : \operatorname{seq} CHAR;\ readership : \mathbb{P}\, People; \\ rating : People \pfun 0 \mathinner{..} 10 \mid readership = \operatorname{dom} rating\ ]$$

There are two distinct sections of the schema, divided by the horizontal line (read as "where"). The first section of the schema above the "where" defines its components and their types. The second section below the "where" defines

invariants that hold for the schema components. The schema type for the above schema *Book* is given as:

$$\langle\!|\ author : People;\ title : \mathrm{seq}\, CHAR;\ readership : \mathbb{P}\, People;\ rating : People \pfun 0 \,..\, 10\ |\!\rangle$$

and its values are *bindings* of the form:

$$\langle author \Rrightarrow au_1;\ title \Rrightarrow ti_1;\ readership \Rrightarrow re_1;\ rating \Rrightarrow ra_1 \rangle$$

where au_1, ti_1, re_1 and ra_1 are constants of appropriate types.

Schemas can be *unnamed*, in which case they appear without the heading label. A schema can *extend* another, by including its name in the description. As a convention, a schema which is only meant to be included in another (otherwise also known as the *partial* schema) has a name that begins with the letter Φ. If the schema does not modify the state of the included one, by a convention the included schema name is prefixed with Ξ. If a schema changes the state of the included one, the included schema name is prefixed by Δ. For convenience, *renaming* can be used to change the appearance of a schema. Thus $Book[People/Borg]$ is a schema *Book* with all the occurrences of *People* changed to *Borg*. Operations on a named schema *Book* that change its state denote this by using it in the declaration, with a prefix Δ (delta). The elements of a schema after the change are primed ("′"). As a convention, when a schema describes a change of state, a question mark ("?") is appended to the names of variables providing external input. Likewise, an exclamation mark ("!") is appended to the names of the variables used as output.

AddReaders

$\Delta Book$
$reader? : People$
$reader\_rating? : \mathbb{N}$

$readership' = readership \cup \{reader?\}$
$rating' = rating \cup \{reader \mapsto reader\_rating?\}$

Generic constructs allow families of concepts to be captured in a single definition. An example of a generic concept is the function *first*, from the Z toolkit ([78], page 93), selecting the first element from an ordered pair in which the elements can have arbitrary types X and Y:

$[X, Y]$

$first : X \times Y \to X$

$\forall x : X;\ y : Y \bullet$
$\quad first(x, y) = x$

Recursive type constructions are made through *free types* (from [78], page 82). A free type definition:

$$T ::= c_1 \mid \ldots \mid c_m \mid d_1 \langle\langle E_1[T] \rangle\rangle \mid \ldots \mid d_n \langle\langle E_n[T] \rangle\rangle$$

introduces a new basic type T, and $m+n$ new variables $c_1, \ldots, c_m$ and $d_1, \ldots, d_n$ declared as if by:

$$[\ T\ ]$$

$$\begin{array}{|l} c_1, \ldots, c_m : T \\ d_1 : E_1[T] \rightarrowtail T \\ \vdots \\ d_n : E_n[T] \rightarrowtail T \end{array}$$

where $\_ \rightarrowtail \_$ is x denoting an *injective* function. The "lambda" notation is used to represent an unnamed function as a first-class object. Thus $(\lambda\, a \bullet a + 1)$ is an "incrementor" function object. A function object can be *applied* to obtain a transformation as follows:

$$(\lambda\, a \bullet a + 1)10 = a + 1[a/10] = 10 + 1 = 11.$$

The expression $f(x)[x/a]$ is called the *substitution*, read as: "in $f(x)$, substitute all appearances of x by a."

Sequences of elements of a given type arise often. They are similar to ordered n-tuples in that the order of the elements is important. They differ from the n-tuples in that the length of a sequence is not fixed. For a type T, $\mathrm{seq}\, T$ is the sequence type. A nonempty sequence type is $\mathrm{seq}_1\, T$. Thus if $l, k : \mathrm{seq}\, \mathbb{N}$, then an example legal l sequence is: $l = \langle 1, 2, 3, 4 \rangle$. Another example is $k = \langle 10, 13, 25, 44, 62 \rangle$.

Schema elements can be referred to in Z. If $x \in Book$, then $x.author$ refers to the value of *author* in x. When the object x is clear from the context, the reference to it may be omitted. Likewise, a function is allowed to return a schema *object*. Thus a partial function definition:

$$\begin{array}{|l} library : \mathbb{N} \nrightarrow Book \end{array}$$

is legal. It denotes a library indexing function *library* whose domain is the set of natural numbers and whose range is the set of *Book* schemas. Now it makes sense to talk about $library(1).author$, $library(1).title$ etc.

2.5 The PN and CPN Descriptions

Petri Nets (PNs) are often represented graphically. In Figure 2.1, a producer-consumer model is given in the PN form as an illustration, following closely [69].

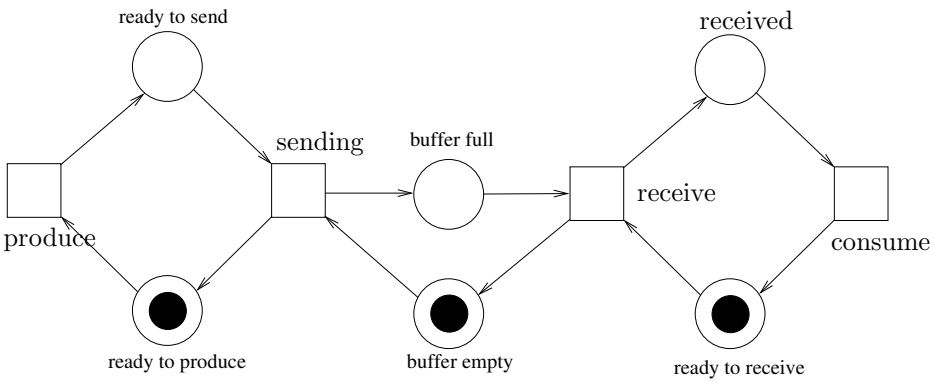


Figure 2.1: A PN representation of a producer-consumer system. The squares are *transitions*, the ovals are *places*, and the arrows are *arcs*. All places have *labels*, and the places with the labels *ready to produce*, *empty* and *receive* have a *token* each.

In its graphical representation, a PN model is an oriented bipartite graph, drawn between nodes denoted as ovals and rectangles. Ovals are named *places*, and rectangles are named *transitions*. The edges of the graph are named *arcs*. Arcs are only permitted to either connect a place to a transition, or vice-versa. No place is connected by an arc to another place, nor is a transition connected to another transition. On each place, a dot can be drawn. The dot is called a *token*. Places and transitions can be marked with a *label*.

The set of all places is denoted as P. The set of all transitions is denoted as T. The set of all arcs is denoted as F and is often called the *flow relation*. The placement of tokens is called the *state*, and a PN is usually given in terms of the token marking for the initial state. The set of transitions that have the arcs pointing to a particular place p of P is denoted as $\circ p$, and the set of transitions pointed to the arcs emanating from p is denoted as $p\circ$. The converse rule holds for a transition t from T. The set of places that have arcs pointing to t is denoted as $\circ t$, and the set of places that are pointed to by arcs emanating from t are denoted as $t\circ$. A transition t is *enabled* if there is a token on all the places from $\circ t$. It *fires* by removing the tokens from $\circ t$ and placing a token on $t\circ$. Firing a single transition is called a *step*. A (possibly infinite) sequence of steps is called an *interleaved run*. In general, more than a single transition can be enabled in a given state, so different interleaved runs can occur.

In Figure 2.1, the producer-consumer system is represented by three circular token flows. These are not specially marked on the figure itself, but by design it is known that the token flow on the left side represents the producer. The token flow in the middle represents the buffer, and the token flow on the right represents the consumer. Initially, the only enabled transition is *produce*, as it is

the only transition t that has a token on all the places $\circ t$. After *produce* fires, a token is placed on *send* and a token is removed from *ready to produce*. Now, the only enabled transition is *sending*, that upon firing removes tokens from *send* and *empty*, and places a token on *full*.

After this transition has fired, there are now *two* enabled transitions in the entire net. These transitions are *produce* and *receive*, so the next transition to fire can be either of the two. Following the token game according to the rules informally outlined here, one is able to construct an interleaved run of the producer-consumer system. To further explain the mechanics of PNs, a formal framework needs to be introduced. The detailed exposition of the framework is given in the book of Reisig [69], and here the most important points of that exposition are highlighted.

Definition 1 (Petri Net) *A Petri Net is a triple $\Sigma = (P, T, F)$, where:*

1. *P is a set of all places;*
2. *T is a set of all transitions;*
3. *F is a set of arcs, or a flow relation for which $F \subseteq (P \times T) \cup (T \times P)$.*

This definition of a PN highlights that a PN is a bipartite graph with oriented edges, as expected from the producer-consumer example. A particular PN is denoted as Σ. When needed, the denotation is indexed by an index of a figure that the referred PN appears on. Thus the net of Figure 2.1 is denoted as: $\Sigma_{2.1}$.

It is likewise easy to define $\circ x$ for $x \in P$, or $x \in T$ as follows.

Definition 2 (Pre- and post- elements) *Let $\Sigma = (P, T, F)$ be a net as in Definition 1, and let $x \in P \cup T$. Define the following sets:*

1. *$\circ x = \{u : \exists (u, x) \in F\}$, and*
2. *$x\circ = \{v : \exists (x, v) \in F\}$.*

In the light of definition 1 and the flow relation F, for $t \in T$, $\circ t \subseteq P$, and $t\circ \subseteq P$. The converse holds for $p \in P$: $\circ p \subseteq T$, and $p\circ \subseteq T$.

The *state* of a PN is defined by the assignment of tokens to places.

Definition 3 (State of a PN) *The state of a PN $\Sigma = (P, T, F)$ is a set $a \subseteq P$. The function: $a : P \to \{0, 1\}$ gives the number of tokens assigned to each place p.*

For $p \in P$, $a(p) = 0$ means that on the place p there is no token for a given state. Conversely, $a(p) = 1$ means that there is a single token on a place p. A PN Σ can have at most one token at a place $p \in P$ in the PN variety from Definition 3. The extensions to this state notation are considered later.

The labeling can be formally defined using the labeling functions as follows.

Definition 4 (Labeling of the PN) *The labeling of a PN $\Sigma = (P, T, F)$ is given by:*

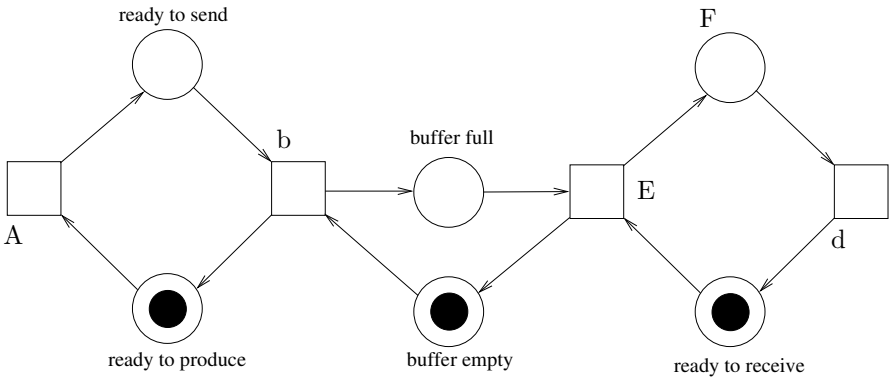


Figure 2.2: A re-labeled net $\Sigma_{2.1}$.

1. *Function* $l_1 : P \to A^*$, *and*
2. *Function* $l_2 : T \to A^*$,

where A^* *is the language over an alphabet* A. *The entire labeling is given by* $l_1 \cup l_2$.

If the state of a PN Σ is given as a, a transition $t \in T$ is enabled in a if $\circ t \subseteq a$. An additional condition is that $t\circ \not\subseteq a$.

Definition 5 *Let* a *be a state of a PN* $\Sigma = (P, T, F)$.

1. *A transition* t *is enabled in* a *if it holds* $\circ t \subseteq a$, *and* $(t \circ \setminus \circ t) \cap a = \varnothing$.
2. *Let* $t \in T$ *be enabled in* a. *The effect of firing the transition* t *from* a, *denoted as* $\mathrm{eff}(a, t)$ *is a state:* $b = \mathrm{eff}(a, t) = (a \setminus \circ t) \cup t\circ$.
3. *Let* $t \in T$ *be enabled in* a, *and let* $b = \mathrm{eff}(a, t)$ *be the effect of firing* t *in* a. *The tuple:* (a, t, b) *is called a* step. *a step is denoted also as:* $a \to_t b$.
4. *For a set of states* $a, a_1, \ldots, a_k$, *and a set of transitions* $t_1, \ldots, t_k$ *such that for each* $i \in \{1, \ldots, k\}$ *the transition* t_i *is enabled in* a_i, *the sequence of steps* $a_1 \to_{t_1} a_2 \to_{t_2} a_3 \cdots \to_{t_{k-1}} a_k$ *is an* interleaved run.

As an example, consider the re-labeled net $\Sigma_{2.2}$. In Figure 2.2 it is given with the initial state $s = \{A, C, E\}$. As $\circ a \in s$, a is enabled in s. The effect of firing a is $q = \mathrm{eff}(s, t) = \{B, C, E\}$, and the corresponding step is $s \to_a q$.

A state formula for a state s of a given PN Σ, is a predicate P that is true in a given state s. It is denoted as: $s \vdash P$. If a predicate P is true for any state s of Σ, it is called a *place invariant* and denoted as: $\Sigma \vdash P$. A predicate is expressed in terms of the state properties of Σ. In the net $\Sigma_{2.2}$, the property of the initial state s is: $s \vdash A \wedge C \wedge E$, which denotes that in state s, there exist tokens on places

A, C, and E. An example place invariant for $\Sigma_{2.2}$ is that there always is either a token on A or a token on B. This observation is expressed by the formula:

$$\Sigma_{2.2} \vdash (A \wedge \neg B) \vee (\neg A \wedge B), \tag{2.1}$$

but in a shorthand notation this is written as:

$$\Sigma_{2.2} \vdash A + B = 1, \tag{2.2}$$

where A and B are shorthands for the values of functions $a(A)$ and $a(B)$ as per definition 3. Three place invariants can be extracted from $\Sigma_{2.2}$, as follows:

$$\begin{aligned} \Sigma_{2.2} \vdash & A + B = 1 \\ & C + D = 1 \\ & E + F = 1, \end{aligned} \tag{2.3}$$

which can be recognized to be, in order, the equations governing the behaviour of the producer, the buffer, and the consumer. This is one of the ways that the functionality expressed by the PN can be mapped to physical entities.

Coloured Petri Net (CPN)

The PN model outlined in the previous sections treats only the so-called Elementary System Nets (ES-nets), in which the execution is determined only by the flow of control (i.e. tokens), and not by the data types. An extension to this model allows tokens that have different types, and allows conditional enabling of the transitions. This model is called the CPN. Just as in the previous sections, only the outline of the model is given here; for complete details the reader is referred again to [69].

A CPN is obtained from a PN Σ, by introducing an *universe* that, for each place $p \in P_\Sigma$, prescribes the set of allowable values of tokens on p, the *universe* of Σ.

Definition 6 (Universe) *The* universe $\mathcal{A}$ *is a mapping from each place* $p \in P_\Sigma$ *to a set* $\mathcal{A}_p$ *of allowable values of tokens in* p*, a* domain *of* Σ*.*

Now, for $p \in P_\Sigma$, and $t \in T_\Sigma$, each arc $f_1 = (p, t)$, $f_2 = (t, p) \in F_\Sigma$ is adorned with an inscription $m(t, p)$ or $m(p, t)$. For each action it holds $m(p, t) \subseteq \mathcal{A}_p$, and $m(t, p) \subseteq \mathcal{A}_p$. The definitions of concession (enabledness) of a transition $t \in T_\Sigma$ is defined analogously to definition 5.

Definition 7 *Let* Σ *be a CPN, and let* a *be its state.*

1. *A transition* t has concession *(is* enabled*) in a state* a *if, for each* $p \in \circ t$ *it holds* $m(p, t) \subseteq a(p)$*.*

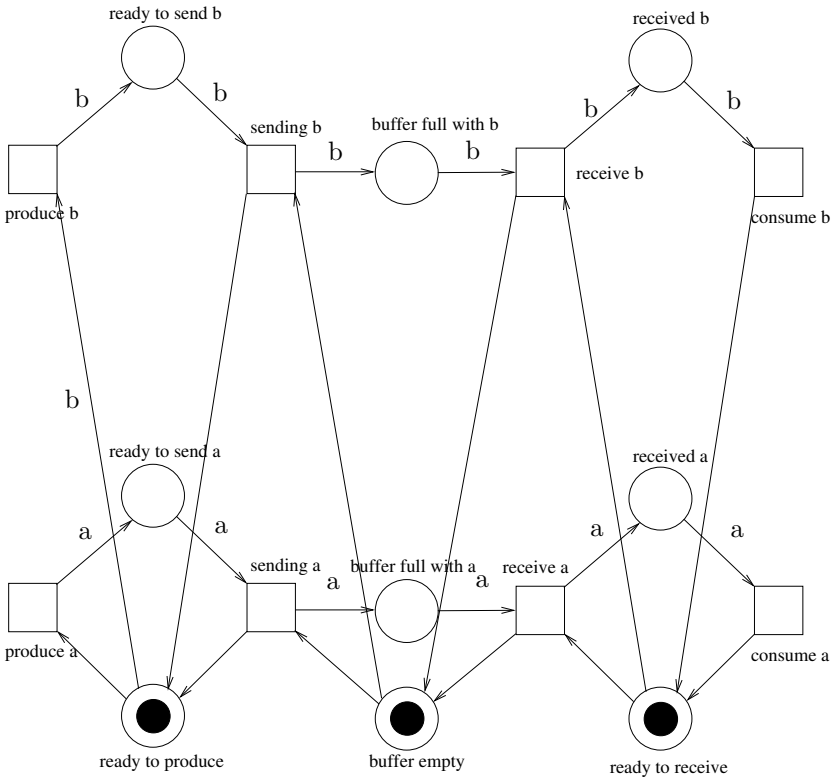


Figure 2.3: An example CPN. This is the advanced version of the producer-consumer net $\Sigma_{2.1}$. A producer can produce either a token a or a token b.

2. *Let $t \in T_\Sigma$ be enabled in a. The effect of firing t in a is the state, for each place $p \in P_\Sigma$: $b(p) = \text{eff}(a,t)(p) = (a(p) \backslash m(p,t)) \cup m(t,p)$, where actions attributed to pairs of p and t without appropriate $f \in F_\Sigma$ are set to: $m(p,t) = \varnothing$ and $m(t,p) = \varnothing$.*

3. *A* step *is defined analogously to that of definition 5.*

4. *The* interleaved run *is defined analogously to that of definition 5.*

As an example for the added functionality, consider the PN in Figure 2.3. In this figure, a producer-consumer pair is again displayed. Now the producer can produce two token types: a and b.

In the ES-net model, the only way to represent this situation is to treat each token production separately. For this reason, separate parts of the net have been constructed for the circulation of a-related tokens (lower part), and the circulation of b-related tokens (upper part). The shared places (*ready*, *empty*, and *ready*

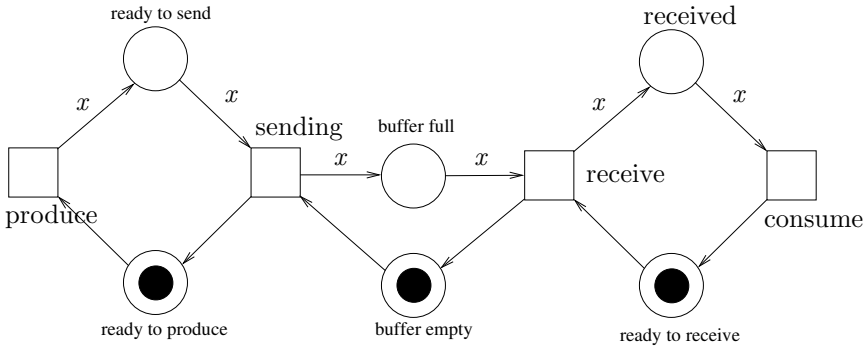


Figure 2.4: A CPN representation of the PN $\Sigma_{2.3}$. The token x inscribed into the arcs can take values from the set $\{a, b\}$.

recv.) now have each a non-deterministic choice of actions to take. The producer determines which action is taken at the beginning of the interleaved run. It can be seen that analogous places exist in the two parts, so that pairs of analogous places exist, with one place intended to track the a token, and the other intended to track the b token. The ES-net models are used in cases where only the token flow, and the firing sequence of the transitions is important. The cases in which also the meaning of each particular token on a place and not only the presence or absence of tokens is important for the activation sequence, gives rise to the so-called Coloured Petri Net (CPN).

The CPN are obtained by identifying analogous places (the places that have similar functionality to some extent). These analogous places can be joined into a single place, with separate markings for the tokens residing there. This operation is called *folding* and can be used to contract the PN model into an equivalent CPN. The ES-net underlying a given CPN model is called an *inscribed net*. For $\Sigma_{2.3}$, the corresponding CPN is shown in Figure 2.4. The folded net shown registers the flow of the token x that can take values from the set $\{a, b\}$. It thus subsumes $\Sigma_{2.3}$. When one refers to a token x at some place A, it is denoted as $A.x$. If there exists a sequence of transitions that, given the presence of tokens $A.x$ and $B.y$ (on places A and B, respectively) fire so that token $C.z$ is produced, one writes:

$$A.x \wedge B.y \hookrightarrow C.z$$

and reads: $A.x$ and $B.y$ *causes* $C.z$. Proof techniques for CPN that take into account the state of the coloured net are analogous to that of the PN as given before. Multiple linked *causes* relations can be expressed in terms of *proof graphs*, which show both the causal relationship and concurrent executions.

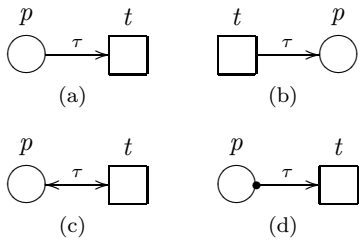


Figure 2.5: The access modes. (a) Removal. (b) Addition. (c) Lookup. (d) Inhibition.

Access Modes

Originally the flow relation of the CPN forms a directed graph over the union of the places and the transitions. The firing of a transitions t means the removal of the corresponding tokens from the incident places $\circ t$, and the production of the corresponding tokens to $t\circ$ (see Figure 2.5). These correspond to two different place *access modes*:

1. *Removal.* The removal access mode at place $p \in$ *Place* for a token $\tau \in$ *Dataspace* deletes τ from the place p (see Figure 2.5a).

2. *Addition.* The addition access mode at a place p for a token τ adds τ to the place p (see Figure 2.5b).

These access modes are denoted by orienting the arrow of each element of the flow relation either away from a place (removal), or towards a place (addition). For practical reasons this semantics of the flow relation is extended to include new access modes. The new access modes are introduced for practical purposes and it is here noted that they may be simulated by using the basic removal and addition modes only. However, for brevity, they are used as follows:

1. *Lookup.* The lookup access mode allows a transition to examine the contents of its incident place in search for a token of particular type. The lookup access mode is denoted by a double-headed arrow between a place and a transition (see Figure 2.5c).

2. *Inhibition.* The inhibition access mode prevents a transition from having concession if the incident place in question contains a token matching a given template. The inhibition access mode is denoted by a dot-tailed arrow connecting the incident place and the transition (see Figure 2.5d).

par

2.6 CPN Simulation by a Blackboard

We turn to the specification of a system given by its CPN description in the Z-with-CPN notation. For this purpose, the CPN description is expanded to include a "universal" data type, called *Dataspace*.

$$[\ \mathit{Dataspace}\ ]$$

The Dataspace type is the union of all the elements that can be obtained by using the *basic types* and a finite number of iterated *aggregations*.

The basic types of the CPN are *Place*, and *Transition*. The precise contents of the types will be specified later. Here they are parachuted into the specification.

$$[\ \mathit{Place}, \mathit{Transition}\ ]$$
$$\mathit{Flow} == (\mathit{Place} \cup \mathit{Transition}) \leftrightarrow (\mathit{Place} \cup \mathit{Transition})$$

The PN itself is defined by defining the triple $(P : \mathit{Place}, T : \mathit{Transition}, F : \mathit{Flow})$.

PN

$$P : \mathbb{P}\ \mathit{Place};\ \ T : \mathbb{P}\ \mathit{Transition};\ \ F : \mathit{Flow}$$

$$\forall\, x, y : \mathit{Place} \cup \mathit{Transition} \bullet$$
$$(x, y) \in F \Rightarrow (x \in P \land y \in T)$$
$$\lor\ (x \in T \land y \in P)$$

The set of transitions that have the arcs pointing to a particular place p of P is denoted as $\circ p$, and the set of transitions pointed to the arcs emanating from p is denoted as $p\circ$. A similar rule holds for a transition t from T. The set of places that have arcs pointing to t is denoted as $\circ t$, and the set of places that are pointed to by arcs emanating from t are denoted as $t\circ$. A transition t is *enabled* if there is a token on all the places from $\circ t$. It *fires* by removing the tokens from $\circ t$ and placing a token on $t\circ$. Firing a single transition is called a *step*. A (possibly infinite) sequence of steps is called an *interleaved run*. In general, more than a single transition can be enabled in a given state, so different interleaved runs can occur.

$$\Xi PN$$
$$\circ\_;\ \_\circ : \mathbb{P}\ \mathit{Place} \cup \mathbb{P}\ \mathit{Transition}$$

$$\forall\, p : \mathit{Place} \in P \bullet$$
$$p\circ = \{t : \mathit{Transition} \mid t \in T \land (p, t) \in F\}$$
$$\circ p = \{t : \mathit{Transition} \mid t \in T \land (t, p) \in F\}$$
$$\forall\, t : \mathit{Transition} \in T \bullet$$
$$t\circ = \{p : \mathit{Place} \mid p \in P \land (t, p) \in F\}$$
$$\circ t = \{p : \mathit{Place} \mid p \in P \land (p, t) \in F\}$$

A CPN is obtained from a $\Sigma : PN$, by introducing *coloring* consisting of an *universe* $\mathcal{A}$ and an *action* m. The universe $\mathcal{A}$, for each place $p \in P_\Sigma$, prescribes the set of allowable values for tokens on p, the *domain* $\mathcal{A}(p)$. On each place, an annotation called a *token* can be drawn. The placement of tokens is called the *state*, and a PN is usually given in terms of the token marking for the initial state. Placeand transitions can be marked with a *label*.

CPN

$$PN$$
$$\mathcal{A} : Place \to \mathbb{P}\, Dataspace$$
$$m : (Place \cup Transition)^2 \to \mathbb{P}\, Dataspace$$

$$\operatorname{dom} \mathcal{A} = P$$
$$\forall\, x, y : Place \cup Transition \bullet$$
$$(x, y) \in \operatorname{dom} m \Rightarrow (x \in P \land y \in T) \lor (x \in T \land y \in P)$$

The state of a $\Sigma : PN$ is a set $a(p)$, for each $p \in P$, such that $a(p)$ belongs to $\mathcal{A}(p)$. For $p \in P$, $a(p) = \varnothing$ means that on the place p there is no token for a given state. If the state of a PN Σ is given as a, a transition $t \in T$ is enabled in a if $\circ t \subseteq a$. An additional condition is that $t\circ \not\subseteq a$.

$$a : Place \to \mathbb{P}\, Dataspace$$

$$\forall\, \Sigma : CPN \bullet p \in P \Rightarrow a(p) \subseteq \mathcal{A}(p)$$

Now revert to the definition of *Place*. Each $p \in Place$ can contain a subset of tokens from its universe $\mathcal{A}(p)$, depending on the CPN marking given by the state.

Place

$$t : \mathbb{P}\, Dataspace$$

$$t = a(self) \qquad [self \text{ is the instance of } Place]$$

The *Transition* operates on a sequence of objects that pass the *guard condition* corresponding to the concession given to each place of the CPN. This convention binds the two specification languages in Z-with-CPN allowing the precise definition of a concession for each of the places. Every $t : Transition$ specifies a function τ that maps a sequence of acceptable input tokens into a set of the acceptable output tokens.

Transition

$$guard : \operatorname{seq} Dataspace \nrightarrow \{true, false\}$$
$$\tau : \operatorname{seq} Dataspace \nrightarrow \operatorname{seq}(Dataspace \times Place)$$

$$\operatorname{dom} guard \subseteq m(\circ self, self)$$
$$\forall (x : \operatorname{seq} Dataspace, y : \operatorname{seq}(Dataspace \times Place)) \in \operatorname{ran} \tau \bullet$$
$$x \subseteq m(self, self\circ) \land guard(x) = 1 \land y \in self\circ$$

2.7 Blackboard Semantics

The structure handling the modified requirements is the Distributed Blackboard (DB) (see [20]). In this structure, all partial result exchange between the mapped tasks is modeled as communication in terms of data objects, which are fetched from a *data space*, an (abstract) collection of all constructable data objects. The DB as implemented by us is based on top of the Cognitive Agent Architecture (COUGAAR) framework, and then extended to handle the issues outlined here. The DB properties are as follows:

1. *Structure.* The DB is a collection of identical Local Blackboards (LBs), that can hold an unbounded-size collection of distinct objects. Each node (PDA) has access to one local LB.

2. *Modification.* The local LB is modified by either *adding* new objects to it, or *removing* or *modifying* them.

   The addition of new objects to the DB readily implements the *addition* access mode of the CPN (Figure 2.5b). The removal of an existing object from the DB readily implements the *removal* access mode of the CPN (Figure 2.5a). The *modification* of the objects can be implemented twofold. It is either a removal followed by immediate addition, or it is a *lookup* (Figure 2.5c), depending on whether the removal mechanism must be triggered, or not, respectively.

3. *Production.* A node can select a subset of objects in LB for *export*. These objects are called *products* and the incident node is called the *producer*.

4. *Consumption.* A node can select a subset of the product data space to *import*. The incident node is called the *consumer*.

5. *Distribution.* Whenever an object o is added to a local LB such that it is a member of the local *production data space*, the object o is delivered to all consumers whose *consumption data space* contains o.

6. *Annotation.* An object can be annotated by *meta-data*, which are always communicated when an object is transferred between nodes. To emphasize some property P that holds for an object o, we write $o\,\langle P\rangle$. To annotate the object o with a version label "ver = 2", one writes: $o\,\langle \text{ver} = 2\rangle$.

The Static Structure of the Local Blackboard

A LB consists of an object container, the Blackboard (BB), and a number of functional units, the Knowledge Sources (KSs). A KS is activated if an appropriate subset of the data space is contained by the LB. Each activated KS may modify the contents of the LB. The KS activation is asynchronous, so a monitor

must synchronize concurrent LB access. A single LB is a container, holding an arbitrary number of objects from a predefined universe:

$$Blackboard \mathrel{\widehat=} [\ blackboard : \mathbb{P}\, Dataspace\ ]$$

Each KS is a piece of business logic. A KS contains a functional unit, accepting objects as input and producing objects as output:

$$FunctionalUnit \mathrel{\widehat=} [\ f : \operatorname{seq} Dataspace \pfun \operatorname{seq}_1 Dataspace\ ]$$

The objects for $\operatorname{dom} f$ and $\operatorname{ran} f$ are supplied from the LB. $\operatorname{dom} f \equiv \operatorname{seq} Dataspace$ since a self-activated functional unit need not have input parameters. But $\operatorname{ran} f$ is never empty. The parameters for f are obtained from LB, where they are selected by applying a predicate:

$$Predicate \mathrel{\widehat=} [\ execute : Dataspace \to \{true, false\}\ ]$$

The predicate can filter objects from LB to only those interesting for a KS. Three object classes are distinguished: the *added* objects, the *removed* objects and the *changed* objects. The added and removed objects are remembered in a similar schema:

$$AddOrRemoveItem \mathrel{\widehat=} [\ x : Dataspace \mid x \in Blackboard\ ]$$
$$AddItem == AddOrRemoveItem$$
$$RemoveItem == AddOrRemoveItem$$

whereas the change schema describes the object change on the blackboard, so that:

$$ChangeItem == \{\ (x : Dataspace, y : Dataspace) \mid x \in Blackboard, y \notin Blackboard\ \}$$

All change items are grouped in a change set:

$$AddSet == \{\ x : AddItem\ \}$$
$$RemoveSet == \{\ x : RemoveItem\ \}$$
$$ChangeSet == \{\ x : ChangeItem\ \}$$

A *Subscription* collects all the change sets, and uses a *Predicate* to limit its scope only to a subset of LB content for which a *Predicate* evaluates to *true*.

$$Subscription == [\ p : Predicate, a : AddSet, r : RemoveSet, c : ChangeSet\ ]$$

A KS is completely specified by the accompanying *FunctionalUnit*, the *VoidFunction* invoked when a KS is activated, and the set of its *Subscriptions*.

$$[\ VoidFunction\ ]$$

KnowledgeSource

$fu : FunctionalUnit$
$execute : VoidFunction$
$subs : \mathrm{seq}\, Subscription$

In some implementations, such as Cougaar [10], a KS is called a *Plugin* after the name of the software component implementing the KS, so the short alias for a KS is:

$$Plugin == KnowledgeSource$$

The LB system is then assembled from the *Plugins* and the LB.

$$BlackboardSystem \mathrel{\widehat=} [\ bb : Blackboard;\ plugins : \mathbb{P}\, Plugin\ ]$$

The Dynamic Structure of the Local Blackboard

Changes to the blackboard are applied in atomic transactions. The objects of the *AddSet* are added to the LB, with the replacements for the items from the domain of *ChangeSet*. The objects removed from the LB are those marked to be removed (i.e. members of the *RemoveSet* and the members of the domain of *ChangeSet*).

$$Transaction \mathrel{\widehat=} [\ a : AddSet;\ r : RemoveSet;\ c : ChangeSet\ ]$$

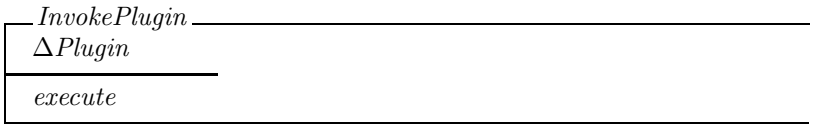
inducing the change in the of the LB contents.

ApplyTransaction

$\Delta BlackboardSystem$
$t? : Transaction$
$to\_remove, to\_add : \mathbb{P}\, Dataspace$

$to\_remove = r? \cup \mathrm{dom}\, c?$
$to\_add = a? \cup \mathrm{ran}\, c?$
$bb' = (bb \backslash to\_remove) \cup to\_add$
$\forall x : Plugin \bullet x \in plugins$
$\quad \forall y : Subscription \bullet y \in x.subs$
$\quad\quad local\_add = \{\ x : Dataspace \mid x \in a?;\ y.predicate(x) = 1\ \}$
$\quad\quad local\_remove = \{\ x : Dataspace \mid x \in r?;\ y.predicate(x) = 1\ \}$
$\quad\quad local\_change = \{\ x : ChangeSet \mid x \in c?;$
$\quad\quad\quad y.predicate(second(x)) = 1\ \}$
$\quad\quad y.a' = local\_add;\ y.r' = local\_remove;\ y.c' = local\_change$

Each *Subscription* is updated by updating its change sets. After these have been formed, each *Plugin* is invoked with the appropriate change set by executing the *VoidFunction*.



2.8 CPN Simulation with a Blackboard

In this section, the CPN and the BB concepts, described in the Sections 2.6 and 2.7 respectively, are brought together to form a fully-fledged toolkit for specifying, describing and implementing distributed systems. The approach presented here is of course but one of many that can be adopted in distributed systems design. The aims of the described toolkit are as follows:

1. *Structuring.* The way that a distributed system component is specified must be expressed in an approved *normal form* which clearly expresses the business logic and the concurrency of the distributed task.

2. *Compilation.* The structured specification must be convertible into an executable system description by an automated process and (possibly) an automated compiler.

3. *Verification.* The structured specification must be amenable to manual or automatic verification.

The toolkit can be viewed as a way of enforcing an engineering discipline into distributed systems design. The discipline we give here is naturally not the only one you could adopt for building own distributed systems, and is certainly not the only one that yields a functional system. However, a disciplined approach to engineering the distributed systems yields understandable designs, which can then be checked against a number of quality-of-service requirements. Such calls for programming discipline have been many, of which we give here some of the most prominent to our finding:

1. Structured programming [26] is based upon the notion that programs can be composed of the fragments, which include statements and several well-defined flow-control structures. Each such fragment can have a single entry point and a single exit point[1].

2. Object-oriented programming is based upon the notion that computer programs are composed of objects that send messages to each other.

3. Design patterns [35] are based upon the notion that certain object instantiations and their connection recur in object-oriented program designs. The

[1]This strict view, due to Dijkstra [26] got an extension which allows the code to have multiple *exit* points. Despite this disagreement about the exact formulation of the structured programming approach, the general principle is nowadays almost exclusively used in practice.

recurrent structures (*patterns*) are promoted as building blocks for well designed programs.

All can be used to *structure* the system design presentation, at the expense of limiting the number of alternative ways to express a particular set of programming actions. The advice given here is comparable in nature to those of structured programming, the object-oriented programming, or the design patterns approach. It is an answer to the question: *What have we learned about building reliable and verifiable distributed systems?*

For the structured distributed system description, the following components are adopted:

1. *Concurrent Events.* The distributed state and the event concurrency are described using the CPN formalism.

2. *State Transitions.* The concurrency description is augmented by the state transitions, described in the Z notation.

3. *Target Architecture.* The target system architecture is a BB system, with the BB semantics as given in the Section 2.7.

The conversion from the CPN description is illustrated with a classical "Dining Philosophers" puzzle, due to Dijkstra[27].

> "Five philosophers, numbered from 0 to 4 are living in a house where the table is laid for them, each philosopher having its own place in the table. Their only problem—besides those of philosophy—is that the dish served is a very difficult kind of spaghetti, that has to be eaten with two forks. There are two forks next to each plate, so that presents no difficulty, as a consequence, however, no two neighbors may be eating simultaneously."

The conversion is presented from the CPN description of the Dining Philosophers problem to an implementation that uses the BB as the target architecture. However, no special handling is provided to guarantee any of the classical constraints[2]. The presented implementation can be extended to incorporate an arbitrary strategy that does coordinate the philosophers' activities and conforms to the classical constraints.

Concurrent Events

The first step in the specification is to determine the set of local states, and the allowed transitions between them. This amounts to the determination of the P,

[2]The classical constraints [19] are: fairness, symmetry, economy, concurrency and boundedness.

T and F components of the PN scheme. In the Dining Philosophers dataspace there are only two types:

$$[\ \mathit{Philosopher}, \mathit{Fork}\ ]$$

so that:

$$\begin{array}{|l} \mathit{Philosopher} = \{\ p_i \mid i \in \{0,1,2,3,4\}\ \} \\ \mathit{Fork} = \{\ f_j \mid j \in \{0,1,2,3,4\}\ \} \end{array}$$

Therefore, the *Dataspace* is in this case:

$$\mathit{Dataspace} == \mathit{Philosopher} \cup \mathit{Fork}$$

As the forks are set between the philosophers so that there is a single fork between each pair of the neighbours at the table, one can define the functions giving the fork being laid left of, or right of, each given philosopher.

$$\begin{array}{|l} l, r : \mathit{Philosopher} \rightarrow \mathit{Fork} \\ \hline l = \{p_0 \mapsto f_0, p_1 \mapsto f_1, p_2 \mapsto f_2, p_3 \mapsto f_3, p_4 \mapsto f_4\} \\ r = \{p_0 \mapsto f_4, p_1 \mapsto f_0, p_2 \mapsto f_1, p_3 \mapsto f_2, p_4 \mapsto f_3\} \end{array}$$

Each philosopher can either be thinking or eating. These states alternate for each given philosopher, although the interleaving of the states across all the philosophers can vary. We can make the philosopher states explicit by adopting places *Thinking* and *Eating*.

$$\begin{array}{|l} \mathit{Thinking}, \mathit{Eating} : \mathit{Place} \end{array}$$

Similarly, a fork can be either available or busy, and the alternation and interleaving are similar as in the case of the philosophers. The states of the forks and the philosophers are not independent, however. If a token $p \in \mathit{Philosopher}$ is in the state *Eating*, the forks $l(p)$ and $r(p)$ must not be in *Available*.

$$\vdash \forall\, p : \mathit{Philosopher} \bullet p \in a(\mathit{Eating}) \Rightarrow l(p) \notin \mathit{Available} \land r(p) \notin \mathit{Available}$$

The converse does not hold, in the sense that the absence of a fork from *Available* does not uniquely define the philosopher:

$$\vdash \forall\, f : \mathit{Fork} \bullet \#l^{\sim}(f) = 2, \#r^{\sim}(f) = 2$$

Therefore a local state representation with three places is in order for fully encoding both the states of the forks and the philosophers. The additional place for the forks is *Available*.

$$\begin{array}{|l} \mathit{Available} : \mathit{Place} \end{array}$$

Each philosopher starts by thinking. He is then taking the forks if available and starting to eat. After a while, he leaves the forks and re-enters the thinking state. Hence two transitions are needed: *Take* and *Leave*.

$$Take, Leave : Transition$$

When defining the flow relation, it is natural to consider that any *Take* transition must involve a previously thinking philosopher and his two adjacent forks. Hence the *Take* must have both the *Thinking* place and the *Available* place as preconditions. The *Take* places the philosopher to the *Eating* state.

A converse process happens once a philosopher stops eating. The ensuing transition is called *Leave*. Thus it must have the *Eating* as its precondition. *Leave* returns the philosopher and the forks back to their original states (*Thinking* and *Available*, respectively). The resulting *PN* is given below.

INIT

PN

$$P = \{Thinking, Eating, Available\}$$
$$T = \{Take, Leave\}$$
$$F = \{\langle Thinking, Take\rangle, \langle Available, Take\rangle, \langle Take, Eating\rangle, \langle Eating, Leave\rangle, \langle Leave, Available\rangle, \langle Leave, Thinking\rangle\}$$

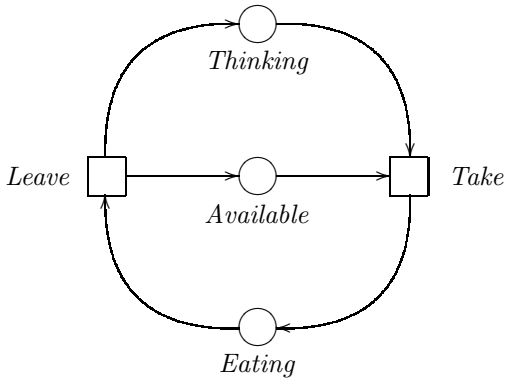


Figure 2.6: The PN fragment that is obtained from considering the places and the transitions only.

The PN given in Figure 2.6 is now coloured to obtain the complete CPN model. According to the exposition of Section 2.6, for this one needs to define the universe set $\mathcal{A}$ and the marking.

CPN_Init

ΞCPN

$\mathcal{A} = \{Thinking \mapsto Philosopher, Available \mapsto Fork, Eating \mapsto Philosopher\}$

In the initial state all the forks are available, and all the philosophers are thinking.

$$
\begin{aligned}
a &= \{Thinking \mapsto \{p_0, p_1, p_2, p_3, p_4\}, Available \mapsto \{f_0, f_1, f_2, f_3, f_4\}, Eating \mapsto \varnothing\} \\
m &= \{\langle Thinking, Take\rangle \mapsto \{x : Philosopher\}, \\
&\quad \langle Available, Take\rangle \mapsto \{l(x), r(x)\}, \\
&\quad \langle Take, Eating\rangle \mapsto \{x : Philosopher\}, \\
&\quad \langle Eating, Leave\rangle \mapsto \{x : Philosopher\}, \\
&\quad \langle Leave, Available\rangle \mapsto \{l(x), r(x)\}, \\
&\quad \langle Leave, Thinking\rangle \mapsto \{x : Philosopher\}\}
\end{aligned}
$$

We obtain the CPN as shown in Figure 2.7.

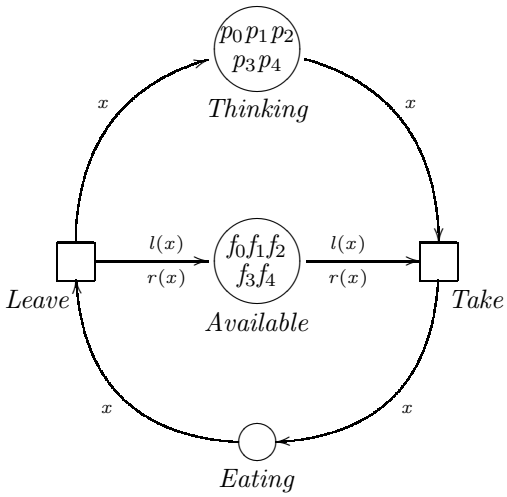


Figure 2.7: The obtained CPN and the initial marking.

State Transitions

The transformations described by each transition can only be triggered if a transition has a concession (i.e. is *enabled*). This occurs for a transition $t : Transition$ when a subset of tokens on $\circ t$ satisfies the *guard* condition of t (see Figure 2.8). For the guard condition of t, the following triggering condition must hold. The

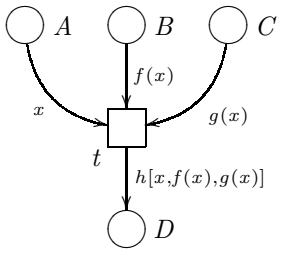


Figure 2.8: Activating a transition $t : \mathit{Transition}$, by the token x from A, and the derived tokens $f(x)$ from B and $g(x)$ from C.

functions f and g are arbitrary functions of the elements of the support set of $A.t$.

$$t \vdash \forall\, x : \mathit{Dataspace} \bullet t.\mathit{guard}[x, f(x), g(x)] = \mathit{true}$$

In the last rule introduces a restriction on the possible sequences of input tokens to a transition. Each token sequence for a transition must be uniquely determined by a token coming from a single place. This token acts as a *key* based on which tokens are selected from the other places as well. This restriction is introduced to enable incremental construction of the guard sequence and is in fact the case which occurs the most in practice. For the illustration purposes, an allowed and a disallowed way to construct the input token sequence is given in Figure 2.9. Firing some transition t causes the change in the local states of $\circ t$ and $t\circ$. For

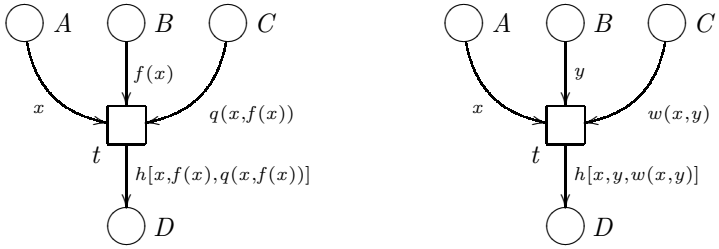


Figure 2.9: The chained determination of the input tokens. (a) Allowed token sequence construction. The token x is obtained from A. Based upon x, $f(x)$ is obtained from B. Finally, the token from C is obtained based on the values obtained from A and B. (b) A disallowed token sequence construction for guard condition testing. Taking arbitrary tokens x and y from A and B respectively is not allowed. The token y must be computable from x directly.

the example of Figure 2.9, the firing of t is given below.

Fire_t

$\Delta A, B, C, D$

$x? : Dataspace$

$A.t' = A.t \backslash \{x?\}$

$B.t' = B.t \backslash \{f(x?)\}$

$C.t' = C.t \backslash \{g(x?)\}$

$D.t' = D.t \cup \{h[x?, f(x?), g(x?)]\}$

Returning to Figure 2.7, the definition of the CPN for the dining philosophers problem is given as follows. For the firing of the *Take* transition:

Take

$\Delta Thinking, Available, Eating$

$x? : Philosopher$

$Thinking.t' = Thinking.t \backslash \{x?\}$

$Available.t' = Available.t \backslash \{l(x?), r(x?)\}$

$Eating.t' = Eating.t \cup \{x?\}$

For the firing of the *Leave* transition:

Leave

$\Delta Thinking, Available, Eating$

$x? : Philosopher$

$Eating.t' = Eating.t \backslash \{x?\}$

$Available.t' = Available.t \backslash \{l(x?), r(x?)\}$

$Thinking.t' = Thinking.t \cup \{x?\}$

Arbiter Transform for Non-Deterministic Transition Firing

In this section we treat the explicit arbiter insertion for the non-deterministic transition firing. In a CPN specification of a distributed process, the issue of non-deterministic choice remains open. The non-deterministic choice arises when multiple transitions are competing for the same token (see Figure 2.10). As far as the CPN semantics is concerned, an arbitrary resolution of the conflict yields a valid run fragment. This property naturally follows the way the CPNs are defined. However it is not immediately useful for the programming as it implies the strong coupling between t_1 and t_2 and cannot be performed in a system based on transactions without external arbitration. Hence in the implementation the arbiter must be explicit. However, the implementations will necessarily differ depending on whether the arbitration is fully contained in a single node, or whether

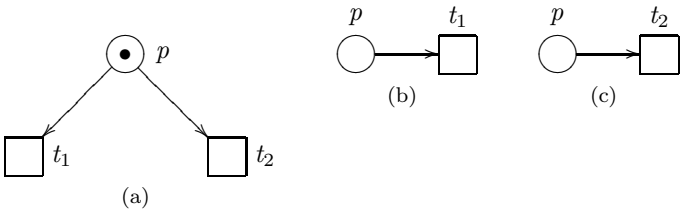


Figure 2.10: The non-deterministic choice on a colourless PN and the respective interleaved runs. The domain of the only place p is just $\mathcal{A}(p) = \{\bullet\}$. (a) The CPN fragment with non-deterministic choice involved. (b) A run with t_1 firing. (c) A run with t_2 firing.

it is distributed. In case of the contained arbitration, usually not more than a selection is enough to direct the tokens to at most one of the enabled transitions. In case of distributed arbitration, provisions must be made so that the arbitration remains manageable, and is not dominated by any participating party. This concern is addressed when a distributed arbiter is made in Chapter 6, as a part of the *Meet* protocol design.

Another example of the non-deterministic decision is the mutual exclusion (*mutex*) primitive. In the most basic mutex setting (see the PN $\Sigma_{2.11}$ in Figure 2.11), there exist two processes $i \in \{1, 2\}$, each of which alternates between a *pending* state (P_i) and a *critical* state (C_i), performing the *enter* (e_i) and *leave* l_i. The mutex requirement states that the processes must not simultaneously be in the critical state. The mutex requirement of $\Sigma_{2.11}$ translates to:

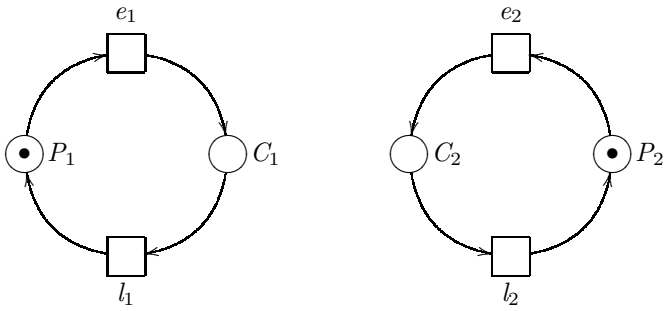


Figure 2.11: The setup for the mutex problem ($\Sigma_{2.11}$).

$$\Sigma_{2.11} \vdash C_1 + C_2 \leq 1$$

Disregarding the non-determinism, the given mutex problem is solved by introducing a shared place M as shown in $\Sigma_{2.12}$. Inserting the place M to the network

$\Sigma_{2.11}$ introduces the invariant:

$$\Sigma_{2.11} \vdash C_1 + C_2 + M = 1$$

from which the required mutex condition is derived:

$$\Sigma_{2.11} \vdash C_1 + C_2 + M = 1 \wedge M \leq 1 \Rightarrow C_1 + C_2 \leq 1$$

The way the contention between e_1 and e_2 is resolved is left unspecified and

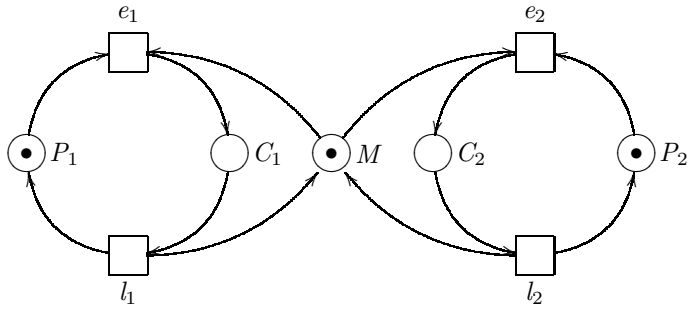


Figure 2.12: The non-deterministic mutex solution to $\Sigma_{2.11}$.

in the PN definition, these ties are broken arbitrarily. Hence no guarantees can be made upon the properties of the tie resolution. This is a hindrance to the implementation where it is oft required that the ties are broken according to some desirable pattern to ensure fairness, for instance. Breaking the ties is left to a device that we here call the *arbiter*. The arbiter decides which of the possible runs is brought about and must be represented explicitly. In the case of $\Sigma_{2.12}$, the arbiter is supposed to be intrinsic to both e_1 and e_2, implying that the two transitions must be coupled. This contradicts the requirement that the transitions e_1 and e_2 must be the parts of independent processes. Hence a first take on the arbiter is to factor the decision out of the transitions e_i (for $i = 1$ and $i = 2$) to yield $\Sigma_{2.13}$.

Target Architecture

Here the conversion of the Z-with-CPN-based description of a distributed system into an executable component is described, such that the description is readily pluggable into an existing distributed system framework. The framework of interest in this writing is COUGAAR. In COUGAAR the complete application is distributed across multiple hosts, where each is capable to execute a number of relatively independent entities of control. These entities are called *agents* in the COUGAAR glossary and we adopt the naming here too.

A COUGAAR agent consists of a number of hierarchically ordered *components*. The components that make up a COUGAAR agent are listed in the specification (called the *society file*), containing a recipe for the assembly of the agent.

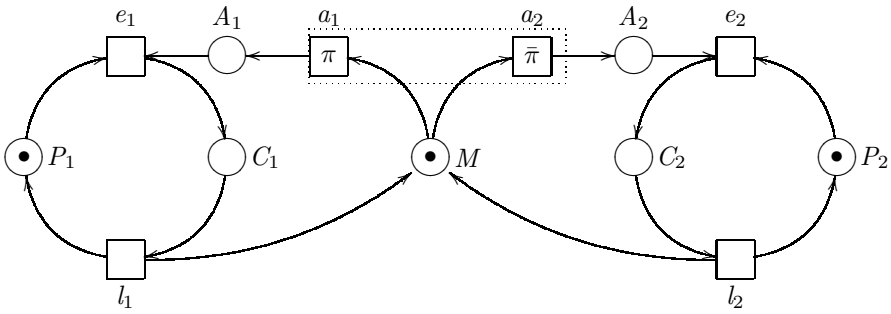


Figure 2.13: Introducing a simple arbiter into $\Sigma_{2.12}$. The arbiter is shown by dotted line. The predicate π and its complement $\bar{\pi}$ are used to decide which process will be granted to the critical section.

The components are bound together by exporting *services*, thereby effectively realizing the Inversion of Control (IoC) pattern (see [10] for the schematic view of the COUGAAR agent components). The most important component for the application writer is the *plugin*. The plugins are able to access the internal agent blackboard and are meant to encompass the agent's main business logic.

When implementing a given Z-with-CPN description of a distributed system, the designer must decide how to allocate and map the portions of the CPN description to plugins. We feel it reasonable to leave the decision to the application author, as there is much freedom of choice in this matter. Our finding is that the functionality typically given in the form of several loosely coupled CPNs should be implemented so that the tightly coupled parts reside in the same plugin.

As the plugins are the containers of the basic business logic of a COUGAAR agent, the plugin structure does not necessarily reflect the logical grouping of the functionality as intended by the application author. In order to allow the functionality grouping on a sub-plugin level, we implemented the container object. The more detailed look at the main ingredients in the implementation of the CPN for COUGAAR can be found in the Figure 2.14.

We now examine the implementation in somewhat more detail. The top level COUGAAR component is the *TransitionContainerPlugin*. This component attaches directly to the COUGAAR infrastructure and, provided that it is correctly initialized with sub-components, performs all the housekeeping needed for the correct initialization of the CPN transitions.

A place in the CPN description must be represented in the implementation by an instance of a class *Place*. A transition must be represented in the implementation by an instance of a class *Transition*. A *Transition* is connected to the container using the following steps:

1. *Activation* or connecting the transition to other transitions by using the *Trigger* objects.

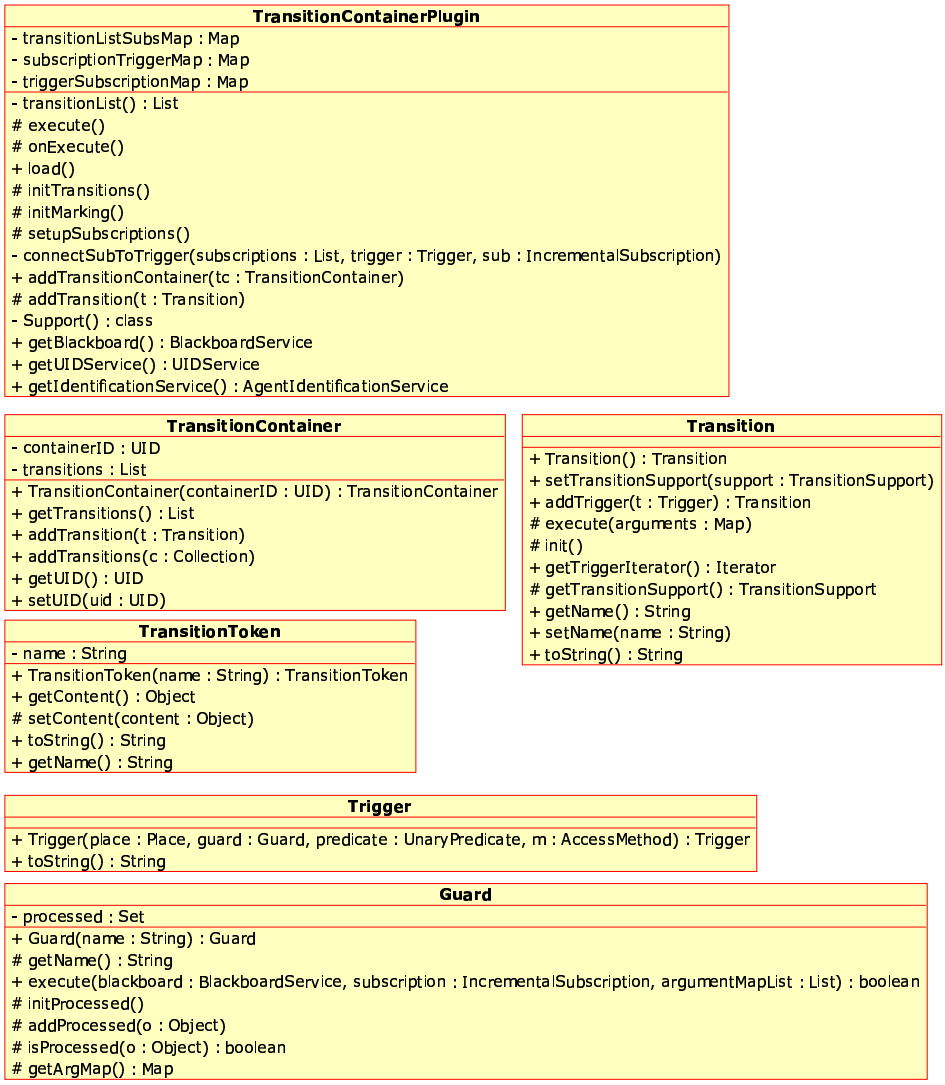


Figure 2.14: The main classes in the implementation of the Z-with-CPN specification on a blackboard-based architecture.

2. *Registration* or inserting into the *TransitionContainer* of choice.

The transition activation, and all the subsequent operations are performed on the transitions and places by *symbolic name*. Hence, a unique symbolic name must be provided for all the places and transitions upon their initialization. This approach has been adopted to allow forward referencing of the places and transitions during the CPN construction. The activation is performed by supplying the *Transition* with a number of *Trigger* objects. A trigger is always bound to some place and requires the following specifications to be present:

1. *Guard* which validates an argument list (i.e. the set of tokens) and, if validated, allows the transition to be invoked;

2. *Predicate* which filters the tokens that come from a particular *place*; and

3. *Access method* which specifies what change to the token set will be relevant for triggering the transition (the access methods were explained in the Section 2.5). The access method can be one of *add*, *remove*, *change*.

A *Guard* is executed each time a *Trigger* detects an access method to a blackboard subset which has been registered with the corresponding *Transition*. Its task is to build the argument list up. It iterates over the interesting places, in sequence, and incrementally builds up a sequence of key-value tuples which will be submitted to the transition when it fires. As the preconditions of the transition are required to be derived from a common *key*, it is possible to build up the argument list by incrementally invoking the registered *Guard* objects for the given *Transition*. Each *Guard* object is then forwarded the partial tuple sequence to which elements are added according to the firing rules. The *Guard* can then either expand the tuple, or it can remove it from the argument list.

Hence, the order of the *Guard* invocation is significant, and the application author must take care to register the guards in the sequence that retains the intended semantics from the Z-with-CPN description.

2.9 Summary

In this Chapter we equipped ourselves with the toolkit required to represent, specify and finally implement the DWEAM system. The presented toolkit will be used extensively in the following Chapters without further elaboration.

Chapter 3

Architecture Overview

This chapter gives the overview of the DWEAM system. The purpose of the exposition is to present in an integrated fashion the important aspects of the DWEAM system and its components. The components themselves will be detailed in future Chapters.

3.1 Introduction

Mobile computers are distinguished from their timely counterparts in that in most applications the outcome of the execution is affected by the influence of the environment. A mobile computer is carried by a user that makes intermittent processing requests. Fulfilling the processing request requires the use of external resources. The simplest use cases are placing a call from a cellular phone, or using General Packet Radio System (GPRS) to access an Internet web-site.

With respect to the computing resource mobility and the type of the networking interconnection, one can identify the following *use cases* (see Table 3.1):

1. With respect to the computing resource mobility:

    a) *Stationary,* where the computing nodes generally remain at designated confined space, such as the home or office desktop;

    b) *Mobile,* where the computing nodes change position in time.

2. With respect to the type of the communication infrastructure:

    a) *Fixed,* where the computing resources use a structurally fixed communication network; and

    b) *Ad-hoc,* where the communication structures are built and maintained at runtime.

Table 3.1: The use cases for the networked computing resources, with respect to the mobility of the computing nodes and the type of the networking infrastructure.

Nodes		Networking		
Fixed	Mobile	Fixed	Ad-Hoc	Example
✓		✓		Home or Office Internet: Web Applications, Multi-Agent Platforms, Distributed Computing, Desktop Groupware
✓			✓	Wireless Home or Office Internet: Web Applications, Desktop Groupware
	✓	✓		Cellular Mobile: Telephone calls, GPRS Internet
	✓		✓	WiFi (Ad-hoc access mode)

With reference to Table 3.1, it is noted that most of the usage modes are well-covered by existing applications. However, the support for mobile computing resources using ad-hoc infrastructure has so far been incomplete.

The ad-hoc access mode of the WiFi[1] protocol allows point-to-point links between wireless nodes to be established. This in effect emulates the medium access policy of the Ethernet. In the Ethernet, arbitrary pairwise connections can be established.

For wireless networks, the pairwise connectivity is achievable in only a limited number of use cases. It stems from the way these networks are constructed. The existence of a wireless connection between two nodes depends on the following factors:

1. *Proximity.* The wireless connection is effected through repeated radio transmissions by the sending end of the connection. The reception of a radio transmission requires that the signal power at the receiving end can be distinguished from any other interfering signal that is received simultaneously. As the signal power decreases, it becomes more difficult to isolate the wanted signal from the interference with the increase in the distance of the communicating end points. Thus, the further away from each other the endpoints are, the more difficult it is for the communication to be established. As a consequence, a given node of a wireless network is likely to be connected only to the nodes in the close proximity.

2. *Interference.* The radio is a shared medium admitting multiple concurrent users. Thus, a given transmission can coexist with several other unrelated but concurrent transmissions. Upon the reception of a given transmission,

[1] WiFi is the common name for the IEEE 802.11 standard

> it must be separated from all the concurrent transmissions which are not of interest. Although the success of the separation schemes varies with the employed separation approach, the overall tendency is that the more powerful the signals from the unrelated transmissions are, the more difficult it is to separate the useful signal.

The mobility of the nodes in the wireless network causes the proximity and the interference to vary. Consequently, it also varies the structure of the wireless network. The number of achievable links varies with time for the Mobile Ad-hoc use case. The variations in the network structure cause disturbances that bubble up to the application level. An application that relies on the known network structure can cease to operate properly.

For practically all the use cases that include stationary nodes and fixed networking, the changes in the network structure happen so infrequently that they are approximated out of the network model. Thus, also a clean separation of concern between the networking and the application layers is achieved. As long as this approximation is valid, the performance of the application built on top of the fixed networking models is satisfactory.

The wireless networks unfortunately break this assumption. Therefore the applications intended for the fixed networking do not carry over into the domain of ad-hoc networks with success. This phenomenon is characterized in [89] where the first point in the analysis of the open problems in the wireless network realm is:

> "Architectures have been primarily driven by a 'point-to-point' philosophy; we need to better understand a network viewpoint wherein nodes can cooperate intelligently taking advantage of the special properties inherent in wireless communication."

The following approaches to resolve the issue are in order:

1. *Emulation.* In this approach, the algorithms of the ad-hoc networking layer are designed so as to emulate the behavior of a fixed networking layer. The advantage of this approach is that for as long as the emulation is successful, the system components at the upper layers can remain unmodified. The disadvantage is that the emulation may impose severe constraints on the use-case for the emulated network. A typical example of this is the GSM network. While it is based on radio communication, the only allowed use-case is that where all the users are separated by only a single hop from the base-station that provides fixed infrastructure access.

2. *Re-engineering.* In this approach, novel primitives are designed to accommodate the novel use-case. The advantage of this approach is that it potentially opens doors to a range of new applications, which were previously not considered due to the limitations of the respective use-cases. The disadvantage

is that the new approach may introduce incompatibilities with the existing set of system components and design techniques.

In this work we take the second approach, assuming that the benefits of the new use-cases outweigh the disadvantages. Bearing in mind that this new approach requires new system components and design techniques to be formulated, in this chapter we give an integrated account of the designed architecture. The presentation is intended to familiarize the reader with the features of the architecture.

3.2 Resources

Two resource types are considered: the computation resources (also: computers, nodes) and the communication resources (also: network).

Computation resources

The computation resources (nodes) considered here are general purpose *computers*.

Each *node* can be distinguished from others nodes by an unique *message address*.

$[\ MessageAddress\ ]$

$\Phi Node$

$$address : MessageAddress$$

$$\forall\, n : Node \bullet n \neq this \Rightarrow n.address \neq address \quad [\text{Unique address}]$$

Communication resources (network)

The communication resources are used to exchange message objects between resources. The communication have a limited physical range, which depends on the position of the nodes in physical space. Each node can communicate directly only with a comparatively small number of neighbours. The notion of *neighbourhood* here coincides to some extent with the nodes which are physically close to each other. More accurately, it depends on the properties of the communication medium (radio) which depends on mutual distance, but also varies in time.

Although the nodes are equipped with unique message addresses, the addresses can only be used to send message objects between nodes that can communicate directly. Hence it is only useful to distinguish between neighbours. This may not be always possible as the connection environment varies in time.

[Time]

We use the symbol *now* to denote the "present time". The details of the treatment of time are given in Section 4.4, on page 72.

$$now : \text{Time}$$

We assume that the reachability is always symmetric.

$$direct\_link : Node \times Node \times \text{Time}$$

---

messages cannot be sent between $n : Node$ and $m : Node$ at now
$\Rightarrow (n, m, now) \notin direct\_link$

$\forall\, n : Node, m : Node \bullet$
$(n, m, now) \in direct\_link \Rightarrow (m, n, now) \in direct\_link$

Most of the time we are interested only in the direct reachability at current time (now).

$$\underline{direct\_link}_{now} : Node \leftrightarrow Node$$

$$\underline{direct\_link}_{now} = \{\ (n : Node, m : Node) \mid (n, m, now) \in direct\_link\ \}$$

3.3 Layering

The system architecture is organized into four basic *layers*. Each of the layers in its own right consists of a number of *components* that define its functionality. For a diagram of the component inter-dependencies please refer to the Table 3.2. The layers are arranged from top to bottom. They bridge the gap from the user, and the need that the user wants the system to fulfill, to the technology used to fulfill these needs. Hence the topmost layers are problem domain specific, and the bottom layers are technology specific. Only the adjacent layers interact, as usual. The layer components are almost perfectly neatly stacked, and their relations are similar to the relations of the adjacent layers: only the adjacent components can interact.

1. *Application Layer (APL).* The APL is the top-level system layer. Here we specify which computational tasks we require the system to perform.

   After the task has been specified, we define a sequence of systematic transforms that convert a task specification first into a workflow structure that shows explicitly the *decompositions* of a task into subtasks, and the pairwise *precedence* relation between the subtasks.

   Then the workflow is cast into an executable schema that is based on the CPN framework which describes the process concurrency, supplemented with the language based on Z[2] [78] to express the transformation of the local process state. The executable schema is then mapped to a collection of computing resources that can execute it.

[2]Both CPN and Z are explained where appropriate.

2. *Blackboard Logic Layer (BLL).* The BLL layer contains the components that resulted from the allocation of the executable schema to the collection of computing resources.

3. *Connectivity Layer (COL).* The COL layer contains the components that enable the DWEAM system to fulfill its main functionality. This layer provides an infrastructure for the synchronization of distributed computing resources, as well as the token persistence needed for the extended preservation of the system soft state.

4. *Infrastructure Layer (INL).* The INL layer contains the components which provide the basic connectivity structure to the DWEAM system.

   It starts from the maintenance of the elementary connectivity, in which each host has knowledge only of a small number of immediate neighbours. It then builds a minimal communication structure allowing multi-hop connectivity between the hosts.

   On top of this structure, a *matching* is performed. The matching provides the *service discovery*, which is used to determine the required interconnections between the hosts.

Table 3.2: The layering and component structure of the DWEAM system. The layering is presented from the topmost (APL), to the bottommost (INL). Towards the top, the layers are problem-domain oriented, and towards the bottom the layers are technology-domain oriented. The layer components are stacked according to their orientation within the layers. The ENS and PE components both provide a foothold for ENS and both use the SP component.

Layer	Components
Application Layer (APL)	Static Task Structure (STS) Workflow Structure (WS) Petri Net Mapping (PNM) Host Mapping (HM)
Blackboard Logic Layer (BLL)	Logic Component (LOC) Blackboard (BB)
Connectivity Layer (COL)	Event Notification System (ENS) Persistency Component (PE) Matching (MAT)
Infrastructure Layer (INL)	Specification (SP) Connectivity (CONN) Neighbourhood (NG)

3.4 Componentized Layer Structure

Application Layer (APL)

1. *Static Task Structure (STS).* The STS describes the problem that needs to be solved for the user. The STS description must define the *expansion*, describing whether a task can be sub-divided, and if so what the precedence between the subtasks can be.

2. *Workflow Structure (WS).* The WS describes the dynamics of the task execution, detailing the partial results that are passed between the inter-dependent tasks, and the actions executed on the individual hosts upon detecting the presence suitable sets of partial results.

3. *Petri Net Mapping (PNM).* The PNM describes the steps to be performed in order to transform the WS into a schema (based on CPN and Z) that shows the explicit relations between the concurrent activities and defines the precise semantics that govern their execution.

4. *Host Mapping (HM).* Finally, the HM describes the assignment of the concurrent activities to computing resources that execute them.

Blackboard Logic Layer (BLL)

1. *Logic Component (LOC).* The LOC component is made up from individual parts that encompass the application logic of a single host for the computing resources obtained from the APL. These parts are called *plugins*, as they are typically self-contained which "plug into" the Blackboard Logic Layer (BLL) framework.

2. *Blackboard (BB).* The BB is used to bind the parts of the LOC component together in a consistent whole. It mandates the cooperation through an event-based mechanism of subscriptions and callback functions.

Connectivity Layer (COL)

1. *Event Notification System (ENS).* The ENS describes the mechanism by which the distributed computing resources can achieve partial synchronization at runtime. The ENS allows the computing resources to exchange message objects (interchangeably: *events*) in order to define their interaction in detail. The ENS discriminates between events by the event description supplied in the event object itself.

2. *Persistency Component (PE).* The PE provides the facilities that allow the distributed computing resources to store their internal state as fragments sprinkled throughout the distributed society. This state can be recovered upon request whenever possible.

3. *Matching (MAT).* The MAT is used to infer the resources which need to communicate the partial results of their tasks to each other. In order to do this, the resources register a specification of the portion of the dataspace interesting for a particular resource, and the matching is performed through determining which specifications have nonempty intersections.

Infrastructure Layer (INL)

1. *Specification (SP).* The MAT component depends on the SP mechanism to operate. The SP provides a language admitting a simple description of dataspace subsets. By providing suitable set operations on dataspace subsets described by the use of SP, the MAT component is able to determine which computing resources need to establish a communication infrastructure.

2. *Connectivity (CONN).* The CONN component assembles the basic NG-based connectivity and builds an elemental communication structure that guarantees the minimum connectivity between any computing resources for which a connection can be found by following a sequence of NG connections.

3. *Neighbourhood (NG).* The NG component maintains a local list of all the computing resources which are directly reachable from some resource in particular. Resources which do not see each other directly must to use the CONN for cooperation.

3.5 Component Overview

Here we give the specification of the DWEAM architecture components. The presentation of the components is bottom-up.

Neighbourhood (NG)

The NG component is installed at each node. The NG component maintains a set of neighbouring nodes as follows.

$\Phi^2 Node$

$\Phi Node$

$neighbours : \text{Time} \rightarrow \mathbb{P}\, Node$

$\forall\, t : \text{Time}, n : Node \bullet (this, n, t) \in direct\_link \Rightarrow n \in neighbours(t)$

Connectivity (CONN)

The CONN component builds a minimal *connectivity structure* across all the nodes. The connectivity structure is a collection of mutually synchronized local states at all the nodes, such that the Connectivity property is satisfied (see Property 1 below).

The CONN first component that makes claims about the state of all the nodes. A pair of nodes that are not linked directly can still be reachable, if one can make a sequence of connected direct links between nodes.

$$\underline{reachable} : Node \leftrightarrow Node$$

$$\forall\, m : Node, n : Node \bullet (m, n) \in \underline{reachable} \Rightarrow (m, n) \in \underline{direct\_link}^{+}_{now}$$

The connectivity structure maintains the following invariant property.

Property 1 (Connectivity) *Let m and n be nodes. Let m $\underline{reachable}$ n hold. The connectivity structure must provide that a message object sent from m, and having n as the destination, can reach n within finite time from the moment it is sent.*

In order to simplify the management of the CONN component, we require that the connectivity structure also be *minimal*.

Property 2 (Minimal Connectivity) *The connectivity structure must be minimal. No subset of the connectivity structure satisfies the Connectivity property.*

The structure that satisfies Property 2 is called the Core Based Tree (CBT). The CBT is a compound structure distributed over the nodes, and the components of the structure are called *uplinks*.

Uplink

$$address : MessageAddress$$

The cardinalities of the sets of uplinks and downlinks are upper bounded by the set of current neighbours. At most one uplink per node may exist.

$\Phi NodeLinks$

$$uplinks : \mathbb{P}\, Uplink$$

$$\#uplinks \leq 1$$

$$\forall\, u : Uplink \bullet \exists\, n : Node \bullet$$
$$n \in neighbours \land u \in uplinks \land u.address = n.address$$

The uplink sets are synchronized between the neighbours. If a node has an uplink with an address of the neighbour, the corresponding neighbour must not have an uplink pointing the opposite way.

Property 3 (CBT) *The CBT is either a:*

1. *Minimal weakly-connected directed graph on a set of all nodes and the set of all uplinks; or a minimal weakly-connected directed unicycle*[3] *on a set of all nodes and the set of all uplinks.*

2. *If the CBT is a tree, there is an unique node without an uplink, the* core*. If the CBT is a unicycle, there is no core, the cycle subset of the CBT is strongly connected.*

As mandated by the Property 3 and suggested by its name, the CBT resembles either an *oriented tree*, or an *oriented unicycle*.

When two CBTs come into contact, they must be joined into a single CBT. The Property 3 holds for the resulting CBT. Since it takes time for the join operation to complete, it is allowed that the CBT Property be temporarily violated. The frame in which this is allowed is called the *Join transaction*. For the join transaction the following invariant holds.

Property 4 (CBT Join Transaction) *For the join transaction for two CBTs:*

1. *During the transaction, Property 3 is satisfied in both CBTs;*

2. *After the join, Property 3 is satisfied for the resulting CBT.*

Specification (SP)

The SP component defines the data modelthe DWEAM system. The SP consists of sub-components:

1. *Dataspace (DS).* The DS is the space spanned by the constructable data types.

2. *Specification (SP).* The SP is a compact representation of the DS subsets.

Dataspace (DS)

The DS is informally described the set of all constructable types. A brief account of the construction is given here, while Chapter 7 contains the detailed explanation. *Atoms* are the common machine-dependent types.

$$Atoms \ ::= \ Naturals\langle\langle\mathbb{Z}\rangle\rangle \mid Reals\langle\langle\mathbb{R}\rangle\rangle \mid Strings\langle\langle \text{seq } CHAR\rangle\rangle$$

Constructors are used first to compose aggregate objects from the basic ones. The *SchemaDefs* represents the aggregate schema types and the *Compounds* are

[3] *A* unicycle *with a unique closed path (i.e. cycle).*

combinations of elements defined previously.

$$[\ SchemaDefs\ ]$$
$$Compounds\ ::=\ Sets\langle\langle \mathbb{P}\ Compounds \cup Atoms \rangle\rangle$$
$$\mid Sequences\langle\langle \mathrm{seq}\ Compounds \cup Atoms \rangle\rangle$$
$$\mid Schemes\langle\langle SchemaDefs \rangle\rangle$$

Next the schema definitions are given with *SchemaDefs*.

$$SchemaDefs\ ::=\ SingleSchema\langle\langle\ (\!|\ id : Dataspace\ |\!)[id] \rangle\rangle$$
$$\mid MultiSchema\langle\langle (\!|\ id : Dataspace\ |\!)[id] \times SchemaDefs \rangle\rangle$$

Finally, the above components can be combined, completing the specification of the *Dataspace*.

$$Dataspace\ ::=\ Atoms \mid Compounds \mid SchemaDefs.$$

Specification (SP)

The SP is used to represent portions of the DS. The SP summary is used to pin down the DS context to which thereafter an operation can be applied. A number of operations on SP descriptions are defined, which admit the formation of compound summaries.

The summaries are used in the matching algorithm for finding the compatible producers and consumers.

$$[\ Type\ ]$$

$$[\ CHAR\ ]$$
$$Name == \mathrm{seq}_1\ CHAR$$

The point in a DS is determined by specifying *coordinates*. Following the overloading concepts from object-oriented languages, we allow multiple coordinates with the same label. However, in order to distinguish between identically labeled coordinates, a second attribute, the *type* is introduced.

$$[\ Type\ ]$$

Textual representations of the types are all of the type *Name*.

$$Name == \mathrm{seq}_1\ CHAR$$

A coordinate is then defined by its label and its type.

$$Coordinate \mathrel{\widehat=} [\ name : Name;\ type : Type\ ]$$

Each *Coordinate* can take on any value from its support type. If we want to consider only parts of the entire support set (denoted as X), a constraint can be specified.

$$Constraint[X] == \mathbb{P}\, X$$

Specifying a constraint for some coordinate yields a *surface*. Multiple surfaces on the components of the same type X taken together, comprise a *cube*.

$$Surface == Coordinate \times Constraint$$
$$Cube == \mathbb{P}\, Surface$$
$$CompoundCube == \mathbb{P}\, Cube$$

The cube membership of a *Dataspace* point is defined through the use of the reflection operators.

$$\_ \in \_ : Dataspace \leftrightarrow Cube$$
$$\_ \in \_ : Dataspace \leftrightarrow CompoundCube$$

The required set operations are defined on cubes, and cube sets. Their definition overloads the generic operations:

$$\_ + \_ == \_ \cup \_$$
$$\_ \cdot \_ == \_ \cap \_$$
$$\_ \cdot \_ : Cube \times Cube \rightarrow Cube$$
$$\_ \cdot \_ : Cube \times CubeSet \rightarrow CubeSet$$
$$\_ \cdot \_ : CubeSet \times CubeSet \rightarrow CubeSet$$

The *sharp operators* [15] are defined in order to support the set operations on cubes.

$$[X, Y]$$
$$\_ \# \_ : Cube[X] \times Cube[Y] \rightarrow CubeSet$$
$$\_ \circledast \_ : Cube[X] \times Cube[Y] \rightarrow CubeSet$$

Analogous operations are valid for cube sets too.

$$\_ \circledast \_ : CubeSet^2 \rightarrow CubeSet$$

The availability of sharp operations enables the definition of the final, union operator for cube sets.

$$\_ + \_ : CubeSet^2 \rightarrow CubeSet$$

All these operations are subsequently used in the (approximate) matching algorithm. At this point we omit the algorithm details, as its presentation contains several fine-grained details. The entire algorithm can be found in Chapter 7, on page 145.

3.6 Summary

In this Chapter we presented the DWEAM architecture in a nutshell. That is, we described the relevant properties of the used computation and communication resources. Thereafter we moved on to the exposition of the system layers, and within each layer their specific components. All the layer components are then exposed in brief, as the details of all of them are given in the Chapters that follow.

Chapter 4

Task Mapping

In this chapter, we describe the *task mapping* for DWEAM. Task mapping is the name for a process that transforms the application requirements represented by a set of informally specified tasks into a specification that is executable in a distributed, dynamic computing environment. The steps required to perform the task mapping are collectively referred to as Application Layer (APL).

The APL we describe is designed specifically (but not limited to) the applications in emergency rescue operations, where the merits of the approach can be fully appreciated. It can be thought of as an enabler for a crisis management groupware. The applications involve teams of individuals that need to act in coordination towards a common goal, and who use portable computers for exchanging relevant information. While this is commonplace in command centers, the support for on-site communication leaves much to be desired. The presented design offers a tool-chest that addresses this issue.

4.1 Introduction

What follows is a description of the APL design for a particular kind of a distributed computing system. This APL addresses the issues arising when a distributed computing system is deployed in an adverse environment. An adverse environment is such that it can have influence the course of the task execution, by shutting down the computational resources and changing the properties of the network that connects the computers together.

A *distributed computer system* is a collection of connected *computing resources*. These resources execute coupled tasks and exchange intermediate results through a pre-installed network. Nowadays they come in a variety of packages—ranging from mainframe systems, through the familiar desktop computers, to the currently under-represented but ever more penetrating portable devices as PDAs.

DWEAM is designed for the applications in emergency rescue operations. It is assumed throughout this article that a set of computational tasks suitable for this

purpose has been pre-defined and is executing concurrently as an aid to the main task allocated to the emergency teams. The helper tasks can, for instance, be used to report continuously of the progress of the main task, to request assistance or supplies and report findings. Through the influence of the operating environment, the execution of these tasks can be affected. A task allotted to a single computer can fail if the computer is destroyed or detached from the network so that it is unable to communicate its data to the other computers in the network. This is in contrast to decision support information systems as deployed in command centers, where better guarantees on the reliability of the system and communication can be assumed.

The task mapping we present prepares the tasks for the execution on a computer system that can cope with the influence of the environment and can execute the allotted tasks in a favourable manner (this informal description will formalized later in the text). Ideally, one wants to find a computational structure that ensures reliable and efficient computation regardless of the possible intermittent system faults. However, the achievability of this ideal goal is coupled to fundamental impossibility results in distributed computing studies: in distributed computer systems that do not execute tasks in lock step, it is impossible to distinguish a failed task from the one that is executing very slowly ([43]). Likewise, in face of communication failures ([51], Theorem 5.1). This result has far reaching implications for the system architecture, as it constrains the assistance to the response team that it can offer. Consequently, we are forced to relax the idealized requirement to meet realistic goals.

4.2 Requirements

The goal of the task mapping is to transform the desired system behaviour (expressed by the set of *tasks* that we want it to perform) into *business logic components*[1] executable by the available resources. The tasks of particular interest are those supplemental to crisis management operations. It is assumed that the crisis management task is executed by a number of (personnel) units, that may be coupled by some kind of command chain. The structure of the command chain is not relevant for this exposition. However, the support offered by the described system must be generic enough to accommodate the existing command and control methods.

In the following sections, we derive the distribution structure for the computational tasks, starting from informal descriptions of prototypical tasks in crisis management. This is to motivate input from the crisis management research community to supply other appropriate applications. Thereafter we zoom into computational tasks within the crisis management domain and describe the STS. The dynamic task structure follows, and converted to CPN to structure it fully.

[1] That is, the executable program code which is structured so to be insertable as a component into a pre-defined software framework.

The section on task mapping gives the method to assign tasks to computers while preserving the task structure. The section on design constraints explains what threats prevent the maintenance of correct task structure, with our solution and performance trade-offs. The section on distributed blackboard describes the structure that supports the given requirements. A survey of related work follows, explaining the origins of the ideas we used in our exposition, and thereafter the conclusion is given.

Task Structure

We envision a near-future search-and-rescue operation, in which members of the participating emergency service units (police, fire brigade, ambulance, etc.) are equipped with PDA devices (*nodes*), each capable of short range radio communication. These devices are used to exchange messages: findings, request backup, distribute orders etc. Only the devices within radio range can directly contact each other. Devices which are connected indirectly, through successive connections, rely on their neighbours to forward the messages. The network of nodes is constructed based on the connections created by a short range radio network: two nodes are connected if they are closer to each other than a pre-determined *range*. This assumption is shared by papers on wireless network properties, such as [39].

We adopted a number of example *operational tasks*, to illustrate the type of work that is expected of the system to perform.

1. *The distribution of orders down the command chain.* In this example task, the system must support the distribution of orders. We assume a hierarchical ordering of the available units, whereby the hierarchy is established in terms of the command chain. The order is a data item (or: data object), generated at a single unit in the hierarchy, that must be distributed to all the units that are its offsprings in the hierarchy tree.

2. *The convergence of the order results up the command chain.* In this example task, the system consists of a number of *sensors* emitting readings about the environment. The convergence is the feedback of these readings to a single node where the readings can be processed. The assumptions about the hierarchy is the same as that of the previous task.

3. *Cooperative map building.* In this example task, the system must support the cooperative building of a site map. The entire team starts off with an empty, unannotated terrain map. Each team member can annotate the terrain map, taking note of newly discovered features. In a rescue operation, the annotations can signify sites where casualties are found, where the terrain conditions have changed from their default (due to debris for instance), or where logistic support (supplies) should be delivered. The annotations made by each team member are shared with others.

Static Task Structure (STS)

The tasks are twofold: the *operational tasks* are performed by *actors* (humans), and *computational tasks* that are performed by *agents* (computers). For the rest of this exposition, only the computational tasks will be specially considered; we assume that a clean cut can be made to separate the task types, and that they are easily conjoined when the system is deployed.

All the tasks require a coordinated set of actions to be performed by the team members. By each action, a part of the task (i.e. a subtask) is executed. The execution of an action brings about the subtask, and produces a partial result, which can be shared with the other team members. The coordination is achieved through *synchronization*, whereby different team members agree to execute their actions in a certain order, and through *cooperation*, whereby the results of the outcome of a single action are used to enable a subsequent action. We model coordination by messages exchanged by the team members, where each message contains a data object. A team member pends the action until an activation message is received (e.g. an order arrives to execute an action). When a message is received, its contents are used in the action contents to execute the subtask.

The overall task is achieved by delegating subtasks to team members, and specifying the way in which the subtasks interact with each other. For handling a variety of tasks such as those of the example, we define the STS, with the description thereof relegated to a separate subsection.

Task Description

Tasks are described at mixed levels: informal descriptions given in English are used as needed to motivate formalization; formal symbolic descriptions are used once the task is described well enough to admit symbolic manipulation. The type of the tasks is the opaque type *Tasks*. The computational tasks are application defined. The messages are treated as coming from an abstract *Dataspace*.

$$[Tasks, Dataspace]$$

Emphasize a single, top level task which contains the overall description of what service the system offers. This task is denoted as T where appropriate.

$$T : Task$$

Task Properties

1. *Decomposition.* A given task may be *decomposed* into a set of subtasks. It is assumed that many useful tasks in crisis management can be decomposed in this way. Subtasks may be further decomposed until an elementary (atomic) subtask is reached.

$$TaskExpansionType == Tasks \leftrightarrow Tasks$$

$$\_ \rightarrow \_ : \mathit{TaskExpansionType}$$

An atomic subtask transforms objects by applying a *transfer function* from TTF_.

$$\mathit{TasksToDataspaceFunction} == \mathit{Tasks} \rightarrow \mathbb{P}(\mathit{Dataspace} \rightarrow \mathit{Dataspace})$$

$$\mathrm{TTF}\_ : \mathit{TasksToDataspaceFunction}$$

2. *Activation.* A task is started if a set of activation messages have been received. Additionally, the set of received activation messages must satisfy all the *guard conditions*, which is a predicate on a set of objects. Upon completion, a task produces *partial results*, i.e. messages that can activate other tasks. The set of activation messages ACT_, partial results PART_ and guard conditions GUARDS_ are given by:

$$\mathit{TasksToDataspaceSet} == \mathit{Tasks} \rightarrow \mathbb{P}\,\mathit{Dataspace}$$
$$\mathit{GuardType} == \mathit{Tasks} \rightarrow \mathbb{P}(\mathbb{P}\,\mathit{Dataspace} \rightarrow \{\mathit{true}, \mathit{false}\})$$

$$\mathrm{ACT}\_, \mathrm{PART}\_ : \mathit{TasksToDataspaceSet}$$
$$\mathrm{GUARDS}\_ : \mathit{GuardType}$$

A task which is not a subtask of any other is the *top level task*, and its partial result is, at the same time, the (overall) result of the task execution. The top level task is executed if all of its subtasks have been executed. An elementary task is executed if it has changed the system state according to its specifications.

3. *Precedence.* A precedence relation "$\prec$" (read as: *precedes*) is established for tasks. For two tasks T and U, we denote $T \prec U$ if and only if U is allowed to be executed only after T has been completed.

$$\_ \prec \_ : \mathit{Tasks} \leftrightarrow \mathit{Tasks}$$
$$\forall (x, y) : \mathit{Tasks}^2 \bullet x \prec y \Leftrightarrow \; x \text{ must be completed before } y \text{ starts.}$$

4. *Coupling.* Tasks T and U, where $T \prec U$ are also *coupled*, if the partial result of T is used as the activation message to U.

$$\mathit{TaskRelation} == \mathit{Tasks} \leftrightarrow \mathit{Tasks}$$

$$\_ \prec_1 \_ : \mathit{TaskRelation}$$
$$\forall (x, y) : \mathit{Tasks}^2 \bullet x \prec_1 y \Leftrightarrow (x \prec y) \wedge [\nexists\, z : \mathit{Tasks} \bullet (x \prec z) \wedge (z \prec y)]$$

These coupled tasks are denoted as: $T \prec_1 U$, and for them $\mathrm{PART}\, T \cap \mathrm{ACT}\, U \neq \varnothing$. When the tasks T and U, are coupled, then by definition there exists no task V such that $T \prec V \prec U$. If, for two tasks T and U neither $T \prec U$ nor $U \prec T$, the tasks are *decoupled*, or *concurrent*. For the coupled tasks, the *activation* property holds.

Property 5 (Activation)

$$\vdash \forall\, x : \mathit{Tasks}, y : \mathit{Tasks} \bullet x \prec_1 y \Leftrightarrow \mathrm{PART} x \cap \mathrm{ACT} y \neq \varnothing$$

The activation property expresses the coupling relation of T and U through the fact that the set of the partial results of T intersect the activation set of U.

The STS consists of the top level task T all the task property elements:

StaticTaskStructure

$T : \mathbb{P}\ \mathit{Tasks};\ \_ \rightarrow \_ : \mathit{TaskExpansionType}, \_ \prec_1 \_ : \mathit{TaskRelation};$
$\mathrm{TTF}\_ : \mathit{TasksToDataspaceFunction};\ \mathrm{ACT}\_, \mathrm{PART}\_ : \mathit{TasksToDataspaceSet};$
$\mathrm{GUARDS}\_ : \mathit{GuardType}$

$$\mathit{STS} : \mathit{StaticTaskStructure}$$

Dynamic Task Structure

The dynamic task structure is described in terms of a Petri Net (PN). The PNs give a compact way to model and reason about the concurrent task execution. We first present the PN and the derived Coloured Petri Net (CPN) model, and then define the *EM*, expressing the task structure in terms of an equivalent executable CPN.

4.3 Enabler Mapping (EM)

To transform a set of tasks into an executable structure, the EM is used. The EM converts the STS to an equivalent CPN model, thus turning a static description into an executable schema.

$$\mathit{EM} : \mathit{StaticTaskStructure} \rightarrow \mathit{CPN}$$

The function is total as any *StaticTaskStructure* must be convertible to a CPN. We specify *EM* constructively, in a sequence of steps that builds CPN components from a STS. For convenience, a renamed schema $\mathit{CPN}[T/U]$ is produced instead of *CPN* as symbols T coincide in *StaticTaskStructure* and *CPN*. The renaming

allows to drop the indices and consider only $STS : StaticTaskStructure$, and $\Sigma : CPN[T/U]$, for which the maplet $STS \mapsto \Sigma$ is within EM.

The maplet is built in a sequence of steps that together construct the components of the CPN Σ.

1. *Retrieve atomic tasks.* We need only consider atomic tasks. Non-atomic tasks are complete automatically once all the tasks from their decomposition are complete.

$$atomic : \mathbb{P}\ Tasks$$

2. *Define transitions.* Make a transition for each atomic task. Additionally, attribute the transfer function to each transition.

$$\begin{array}{l} trans : Tasks \nrightarrow Transition \\ trfun : Transition \nrightarrow \mathbb{P}(Dataspace \rightarrow Dataspace) \\ \hline \mathrm{dom}\ trans = atomic \end{array}$$

3. *Define places.* Define a place in P for any two coupled tasks from $\_ \prec_1 \_$.

$$\begin{array}{l} places : Tasks \times Tasks \nrightarrow Place \\ \hline \mathrm{dom}\ places = \_ \prec_1 \_ \end{array}$$

4. *Define the flow.* Define two components of the flow relation. First, the *precondition* component, connecting "inbound" *Place* to *Transition*. Second, the *postcondition* component, connecting *Transition* to "outbound" *Place*.

$$\begin{array}{l} prec : (Tasks \times Tasks) \times Tasks \nrightarrow Flow \\ postc : Tasks \times (Tasks \times Tasks) \nrightarrow Flow \\ \hline \mathrm{dom}\ prec = \{x : Tasks \times Tasks, y : Tasks \mid x \in \_ \prec_1 \_ \wedge y \in \mathrm{dom}\ x\} \\ \mathrm{dom}\ postc = \{x : Tasks, y : Tasks \times Tasks \mid x \in \mathrm{ran}\ y \wedge y \in \_ \prec_1 \_\} \end{array}$$

The flow relation is the union of these two components.

5. *Define the universe.* The universe is borrowed from the activation property for coupled tasks.

6. *Define the inscription.* The inscription determines which transitions are enabled. The inscription on an arc connecting a place and a transition is a set of subsets of points in the data space for which all the guard predicates are valid. The inscription on an arc connecting a transition and a place is a set of subsets of points from the data space.

Theorem 1 (Mapping) *Let there be given a STS describing the static task structure, and let Σ be a CPN. $STS \mapsto \Sigma$ is a mapping if the property set (following the steps given above) is fulfilled.*

$$\Sigma, STS \vdash atomic = ran(\_ \to \_) \setminus dom(\_ \to \_)$$
[1. Retrieve atomic tasks]

$$\Sigma, STS \vdash U = ran\, trans;\ trans = \mathrm{TTF}_{\_} \fatsemi trfun^{\sim}$$
[2. Define transitions]

$$\Sigma, STS \vdash P = ran\, places$$
[3. Define places]

$$\Sigma, STS \vdash F = ran\, prec \cup ran\, postc$$
[4. Define the flow]

$$\Sigma, STS \vdash (x : Place, y : \mathbb{P}\, Dataspace) \in \mathcal{A} \Leftrightarrow \exists\, z, t : Tasks \in T \bullet$$
[5. Define the universe]

$$z \prec_1 t \wedge x = places(z, t) \wedge y = \mathrm{PART}z \cap \mathrm{ACT}t$$

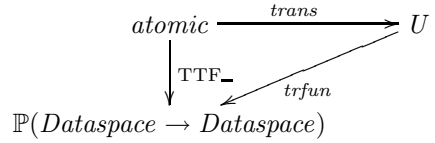
$$\Sigma, STS \vdash \forall\, p : Place, t : Transition \bullet$$
$$(p, t) \in F \Leftrightarrow m(p, t) = \{x : Dataspace \mid x \subseteq \mathcal{A}_p$$
[6. Define the inscription]

$$\wedge\, \exists\, s : \mathbb{P}\, Dataspace \bullet \forall\, g : \mathbb{P}\, Dataspace \to \{true, false\} \bullet$$
$$g \in \mathrm{GUARDS}trans^{\sim} t \wedge x \in s \wedge g(s)\};$$
$$(t, p) \in F \Leftrightarrow m(t, p) = \{x : Dataspace \mid x \subseteq \mathcal{A}_p\}$$

Proof. The proofs of the properties are given in the order they are presented.

1. The case in which an atomic task is a top level task at the same time is uninteresting, as that task is not distributable. Atomic tasks cannot be decomposed, so they cannot appear in $\mathrm{dom}\ \_ \to \_$. However, they must all appear in $\mathrm{ran}\ \_ \to \_$ as each must have been produced by a decomposed task.

2. The following diagram gives the relationships of the functions, their domains and ranges. Note that only the atomic tasks form the transitions in the construction of U.



3. Immediate from the definition.

4. Immediate from the definition.

5. By Property 3, if $x \prec_1 y$, there is a place $places(x, y)$ in P and $places$ is a bijection. By Property 5, the activation set is nonempty. Hence universe $\mathcal{A}$ maps $places(x, y)$ to $\mathrm{PART}x \cap \mathrm{ACT}y$.

6. Immediate from the definition.

□

The properties 1 to 6 define Σ that corresponds to the given STS. The renaming is reversed so that the maplet is type-compatible with *EM*.

Property 6 (Mapping rule) *Let there be given an Enabler Mapping (EM), and let there be given a Static Task Structure (STS), mapping the static task structures into the corresponding CPNs. Let* $\Sigma[U/T]$ *be a CPN obtained by applying the steps from Theorem 1. Then*

$$\vdash (STS \mapsto \Sigma[U/T]) \in EM$$

That is, $STS \mapsto \Sigma[U/T]$ *is a valid EM component.*

Example 3 *A form of this mapping can be found in the usual networked applications. A web browser executes the task:*

$$T == \text{“display a web page”}$$

for which

$$\mathrm{ACT}\, T = \{\textit{Uniform Resource Locator (URL) input by the user}\}$$

and

$$\mathrm{PART}\, T = \{\textit{present the web page contents}\}.$$

Further, T *decomposes as:* $\_ \rightarrow \_ = \{\ (T, T_1), (T, T_2), (T, T_3), (T, T_4)\ \}$*, where:*

$$\begin{aligned} T_1 == &\quad \text{“request page from the server given an URL”} \\ T_2 == &\quad \text{“accept request for page”} \\ T_3 == &\quad \text{“deliver page”} \\ T_4 == &\quad \text{“accept page delivery”}. \end{aligned}$$

In this case $T_1 \prec_1 T_2 \prec_1 T_3 \prec_1 T_4$*. The coupling between* T_1 *and* T_2 *is achieved by setting* $\mathrm{PART}\, T_1 = \mathrm{ACT}\, T_2 = \{o\}$*, where* $o == \{URL\}$*. Other couplings are given by* $\mathrm{PART}\, T_2 = \mathrm{ACT}\, T_3 = \{\textit{request } o\}$*, and* $\mathrm{PART}\, T_3 = \mathrm{ACT}\, T_4 = \{\textit{page for } o\}$*. The transfer functions are given as:* $\mathrm{TTF}\_ = \{\ T_1 \mapsto \tau_1, T_2 \mapsto \tau_2, T_3 \mapsto \tau_3, T_4 \mapsto \tau_4\ \}$*. The function* $\mathrm{GUARDS}\_$ *is empty.*

Assume that there is a $STS : StaticTaskStructrure$ *with properties as given above. Now we find the elements of a corresponding* $\Sigma : CPN[T/U]$*. The Properties referred to here come from the theorem 1.*

$$\Sigma, STS \vdash (\textit{Property 1}) \Rightarrow atomic = \{\ T_1, T_2, T_3, T_4\ \}$$

By Property 2, there should be a transition per transfer function. These are given by:

$$\begin{aligned} &\Sigma, STS \vdash \\ &\quad (\textit{Property 2}) \Rightarrow \\ &\qquad U = ran\, trans = ran\, \mathrm{TTF}\_^{\sim} \fatsemi trans = \\ &\qquad\quad = \{\ t : Transition \mid \exists \tau \in ran \mathrm{TTF}\_ \bullet t = \mathrm{TTF}\_^{\sim} \fatsemi trans(\tau)\} = \\ &\qquad\quad = \{t_1, t_2, t_3, t_4\} \end{aligned}$$

with:

$$\Sigma, STS \vdash trans = \{T_1 \mapsto t_1, T_2 \mapsto t_2, T_3 \mapsto t_3, T_4 \mapsto t_4\}$$

Also:

$$\begin{aligned}\Sigma, STS \vdash & (Property\ 3) \\ & \Rightarrow P = \{places(T_1 \prec_1 T_2), places(T_2 \prec_1 T_3), \\ & places(T_3 \prec_1 T_4)\} = \{p_1, p_2, p_3\}\end{aligned}$$

The flow relation is established using Property 4.

$$\begin{aligned}\Sigma, STS \vdash & (Property\ 4) \Rightarrow \\ & prec = \{((T_1, T_2), T_1) \mapsto f_1, ((T_2, T_3), T_2) \mapsto f_2, ((T_3, T_4), T_4) \mapsto f_3\} \\ & postc = \{(T_2, (T_1, T_2)) \mapsto f_4, (T_3, (T_2, T_3)) \mapsto f_5, (T_4, (T_3, T_4)) \mapsto f_6\} \\ & F = \{\ f_i : Flow \mid i \in 1\,..\,6\ \}\end{aligned}$$

The universe is determined from the activation message set:

$$\begin{aligned}\Sigma, STS \vdash & (Property\ 5) \Rightarrow \mathcal{A} = \{p_1 \mapsto \{o\}, p_2 \mapsto \{\text{“request } o\text{”}\}, \\ & p_3 \mapsto \{\text{“page for } o\text{”}\}\}\end{aligned}$$

and the inscriptions are analogously:

$$\begin{aligned}\Sigma, STS \vdash & (Property\ 6) \Rightarrow \\ & m = \{(t_1, p_1) \mapsto \{o\}, (p_1, t_2) \mapsto \{o\}, \\ & (t_2, p_2) \mapsto \{\text{“request } o\text{”}\}, (p_2, t_3) \mapsto \{\text{“request } o\text{”}\}, \\ & (t_3, p_3) \mapsto \{\text{“page for } o\text{”}\}, (p_3, t_4) \mapsto \{\text{“page for } o\text{”}\}\}\end{aligned}$$

The resulting CPN is shown in Figure 4.1.

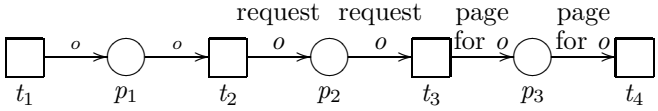


Figure 4.1: The resulting Task Mapping, represented by a CPN.

4.4 Mapping Tasks to Nodes

The main purpose of the task distribution structure is the *task to resources mapping*. The tasks are mapped via the CPN obtained from the Static Task Structure. Two resource types are considered. First, the computational resources (hereafter: *nodes*) which are able to execute the computational tasks. Nodes are hereafter denoted by lower-case Greek letters.

[*Node*]

Second, the communication resources (hereafter: *network*), pairwise connecting some of the nodes. The network is denoted as a collection of links between nodes. The interconnection changes as the nodes move, join or leave the network, so it can be time-variable in general. When nodes leave the network, it can happen either *voluntarily* or *involuntarily*; the latter can occur because of network partitions or nodes being damaged, out of power, or destroyed.

To handle this variability a notion of time is introduced. Only the events that change the global view are relevant. Thus the considered timing is discrete. We are often interested in a specific event for which claims are made, and call it *now*. When considering graph changes, it is useful to consider the "immediate interesting past", so *then* is also reserved. We understand these two symbols as referring to the present moment in time, and some previous event of interest, respectively.

$$\text{Time} == \mathbb{N}$$

$$\begin{array}{|l} now : Time \\ then : Time \end{array}$$

A point $t \in \text{Time}$ is a point at which some event of interest occurred . The real time that separates t from the following event $t+1$ may vary for different choices of t.

$$real\_time == \mathbb{R}$$

$$\begin{array}{|l} RealTime : \text{Time} \rightarrow real\_time \\ \hline \forall\, t, u : \text{Time} \bullet RealTime(t) < RealTime(u) \Rightarrow t < u \end{array}$$

Following the influence of the environment, the number of active nodes can vary over time. The same is true for the active links between the nodes.

$$\begin{array}{|l} active\_nodes : \text{Time} \rightarrow \mathbb{P}\, Node \\ active\_links : \text{Time} \rightarrow \mathbb{P}\, Node \times Node \\ G : \text{Time} \rightarrow \mathbb{P}\, Node \times \mathbb{P}\, Node^2 \\ \hline \forall\, t : \text{Time} \bullet \\ \quad \alpha \in active\_nodes(t) \Rightarrow \\ \quad\quad \text{node } \alpha \text{ is available at} \\ \quad\quad \text{real time interval } (RealTime(t), RealTime(t+1)] \\ \quad (\alpha, \beta) \in active\_links(t) \Rightarrow \\ \quad\quad \text{link from } \alpha \text{ to } \beta \text{ is available} \\ \quad\quad \text{at real time interval } (RealTime(t), RealTime(t+1)] \\ \quad G(t) = (active\_nodes(t), active\_links(t)) \end{array}$$

The nodes and links available at $t : \text{Time}$ naturally induce a graph that is denoted as G_t. For $G(now)$, and appropriate other denotations, indices can be dropped[2].

$$G_t == G(t)$$
$$active\_nodes_t == active\_nodes(t)$$
$$active\_links_t == active\_links(t)$$
$$G == G_{now}$$
$$active\_nodes == active\_nodes_{now};\ active\_links == active\_links_{now}$$

The transition mapping TRANS_ determines which nodes are allocated to which transition. Multiple nodes may implement the same transition.

$$\text{TRANS\_} : \mathit{Transition} \to \mathbb{P}\, \mathit{Node}$$

A transition $t : \mathit{Transition}$ mapped to some $\alpha : \mathit{Node}$, is written as mappedto $t\alpha$. Thus, mappedto $t\alpha$ and mappedto $t\beta$ are instances of the same transition, mapped respectively to nodes α, and β.

The following requirements are imposed on the transition mapping for any $\Sigma : CPN$:

1. *The transition mapping must preserve the CPN structure.* Consider transitions t, and u coupled by a place p:

   $$\Sigma \vdash p = \mu\, q \bullet q \in P \land t \in \circ q \land u \in q\circ$$

   Then, a necessary condition for u to be executed is that the tokens from p are deposited from t to p and then to u. When these transitions are mapped to nodes, such that mappedto $t\alpha$ and mappedto $u\beta$, this means that communication must be ensured between α and β so that the activation of mappedto $u\beta$ is possible.

   $$\Sigma \vdash \exists\, p \Rightarrow (\text{TRANS}t, \text{TRANS}u) \in active\_links$$

2. *The mapping set for each transition must be non-empty.* Every transition $t \in T_\Sigma$ must map to at least one node in the set of active nodes.

   $$\Sigma \vdash \text{TRANS}t \cap active\_nodes \neq \varnothing$$

Ensuring that these two necessary conditions hold is sufficient for the CPN execution[3].

[2] As a salient side effect to the arbitrary choice of the event corresponding to *now*, the properties where items appear with indices dropped must hold for all possible choices of *now*.

[3] However, it is not sufficient to also validate the CPN with respect to safety and liveness properties. Liveness and safety hold only if the CPN has been designed with these properties in mind.

Example 4 *We pick up Example 3 where we left off. A mapping to two nodes is given: active_nodes* $= \{\gamma, \delta\}$*, with active_links* $= \{(\gamma, \delta)\}$*, where* $\gamma ==$ *client and* $\delta ==$ *server.*

$$\text{TRANS\_} = \{t_1 \mapsto \{\gamma\}, t_2 \mapsto \{\delta\}, t_3 \mapsto \{\delta\}, t_4 \mapsto \{\gamma\}\}.$$

It is easy to inspect TRANS_ *to verify that both properties hold.*

Design Constraints

Fulfilling the task mapping requirements is complicated by the dynamic nature of the network formed by the nodes. As the owners of the PDAs move, the structure of the network changes, thus affecting the first transition mapping requirement. Furthermore, as PDAs are subjects to faults and destruction, the second transition mapping requirement can be violated for a task T if all nodes in TRANST are affected.

The loosely-coupled collection of PDAs, as outlined here, matches the asynchronous model of a distributed system being "separate components [taking] steps in an arbitrary order, at arbitrary relative speeds" ([51], page 5). Two important constraints apply for such asynchronous systems:

1. *Coordination is impossible under communication failures.* This is the similar to the "coordinated attack problem" ([51], Theorem 5.1), establishing that in a distributed system, with at least two communicating nodes, it is not possible for the two nodes to agree over a value of a variable[4] if communication can fail.

2. *Slow and stopped nodes are indistinguishable.* This observation ([43]) establishes that nodes in an asynchronous system cannot determine if a particular other node has crashed, or is running slowly.

The given constraints affect the adherence to the transition mapping requirements. We call this influence a *threat*[5].

The first constraint is a threat to the first transition mapping requirement.

Example 5 *Consider transitions* t*, and* u*, and assume that* p *exists as given in the first mapping requirement. The activation of some* u *: Transition depends on the reception of all data objects from the inscription* $m(t, p)$ *from* t*. When these objects are miscommunicated, the nodes* $\alpha =$ TRANSt *and* $\beta =$ TRANSu *cannot agree whether the activation condition is fulfilled:* α *has sent* $m(t, p)$ *so in* α*'s view* u *can be executed. But* β *has not received* $m(p, u)$*, so in* β*'s view,* u *cannot be executed.*

[4]This is known as the "Consensus Problem" (see e.g.[30]). In the consensus problem, two or more computers begin with an initial value of some type, and must eventually output a value of that same type. The outputs of all the computers must be the same.

[5]The term "threat" is taken from AI planning domain ([72], Chapter 11), where it denotes an action that prevents the execution of a formulated partial plan.

Furthermore, as the nodes in G move, join and leave the network, a *transient mapping threat* can occur.

Example 6 (Transient mapping threat) *Let there be two active nodes:*

$$active\_nodes = \{\alpha, \beta\},$$

with transitions t and u, and a place p as given in the first requirement. Let the transition mapping be constant $\text{TRANS}_- = \{t \mapsto \{\alpha\}\}$.

Let active_links be variable, as follows. Assume that there are only three events: $-\infty$, then, and now. These three events divide the real time into three parts. Assume $active\_links_{-\infty} = \{(\alpha, \beta)\}$. At then, the only link is broken so that $active\_links_{then} = \varnothing$, and subsequently restored: $active\_links_{now} = \{(\alpha, \beta)\}$. By inspection one finds that the first requirement is violated in the interval $(0, 1]$.

The second constraint is a threat to the second transition mapping requirement.

Example 7 *It follows that a transition t cannot check whether* $\text{TRANS}t = \varnothing$*, as a probe message sent to a node in* $\text{TRANS}t$ *may receive a reply with unbounded delay.*

In the light of the design constraints, a system that adheres strictly to the task mapping requirements is not achievable. We therefore consider a relaxed, probabilistic formulation of the transition mapping requirements, so that they hold with high probability (*whp*)[6]. We adopt two simple *structuring principles* that achieve this.

1. *Token Persistence.* Instead of limiting the token communication to a certain time frame, thus giving rise to transient mapping threats, the tokens are present indefinitely once generated. In [57] it is shown how this can be achieved *whp*.

2. *Mapping Redundancy.* The TRANS_- must map a transition to multiple nodes in G. The mapping set is shown to be nonempty *whp* in [29].

The structuring principles remove the threats, but introduce side effects, with per-application significance:

1. *Excess Storage.* The indefinite lifetime of the persistent tokens is not achievable in practical systems where the memory is always bounded. This obvious issue is handled in the standard systems by garbage collection.

[6]The term *whp* means that the probability that a requirement holds is a function of a positive parameter n of the form $1 - \frac{1}{n^k}$ for some $k > 1$.

2. *Duplication.* Due to the mapping redundancy, it is possible that multiple, functionally equivalent tokens are produced at runtime. Given a transition t and the mapping TRANS$t = \{\alpha, \beta\}$, transitions mappedto $t\alpha$ and mappedto $t\beta$ produce equivalent partial results. This issue is handled by *visioning* the partial results, so that they can be compared.

3. *Redundant token deliveries.* As a joint effect of token persistence, and the temporal variation of *active_links*, multiple deliveries of the same token, originating from the same node can occur. This amounts to the *at-least-once* message delivery semantics.

4.5 Distributed Blackboard

The Distribution Mechanisms (DMs)

The extension to the Local Blackboard (LB) is the *Export Mechanism (EMX)* allowing the copies of data space objects to be shared across multiple LBs, and thus be locally accessible to the active nodes.

The EMX is a new take on the familiar ideas of *distributed shared memory* and *tuple spaces* (see [3]). The distributed shared memory emulates a unique memory space to which several nodes can have concurrent access. Thus the nodes see the possibly fragmented memory as an unique memory space. In the tuple spaces implementation, the *tuple space* is a domain in which objects exist. An object is instantiated by inserting it into the tuple space. After that it becomes visible for all the nodes attached to the said tuple space. Subsequent operations can modify and delete such an object. The tuple spaces are associated with a *notion of authority* whereby only the resource that has inserted an object can modify it.

In the EMX, a modified approach is considered. Each node maintains an independent local view of the data space. This is achieved by the LB and the local views can be freely modified according to the LB semantics. Cooperation begins once a node marks a subset of its local LB *Exported View (XV)*. The XV is what the node is willing to share with the environment.

In line with the structuring principles introduced earlier, several consequences result.

1. The implied required storage for all the XVs is unbounded, (in line with the excess storage effect);

2. Once exported, an object cannot be deleted or modified (in line with the duplication effect);

3. As each node reports on its XV whenever asked, it is possible that it answers to the same inquiring node multiple times (in line with the redundant token deliveries effect).

Consider all nodes a client to the DB structure. Each client can express an interest in a subset of the dataspace.

$$ClientInterest == \mathbb{P}\, Dataspace$$

Although the client interest is defined as a subset of a possibly large dataspace, it is worth noting that encoding the interest can be much more efficient if only the boundaries of the interest are encoded.

Producer

$$\Xi BlackboardSystem$$
$$exported : ClientInterest$$
$$exported \subseteq blackboard_{bb}$$

The node becomes a *producer* for the subset of the data space covered by *exported*. Any other node is free to ask for a subset of the exported view, making it a *consumer*.

Consumer

$$imported : ClientInterest$$

A node may produce some objects, and consume others (in line of the transition mapping). The node name in this dual role is *Client*.

$$Client == Producer \wedge Consumer$$

The client is free to modify the exports and import interests at will.

ModifyProducer

$$\Delta Producer$$
$$exported? : ClientInterest$$
$$exported' = exported?$$

Likewise for the consumer.

ModifyConsumer

$$\Delta Consumer$$
$$imported? : ClientInterest$$
$$imported' = imported?$$

In the example we compare the conventional view to that of the DB.

Example 8 (Blackboard Bank Account) *Consider a classical example of the concurrent access*[7] *to a project bank account*[8]*. Project participants can use the money from the account to get equipment for the experiments. Alice (A) and Bob (B) each request 1€. The requests are forwarded to the Bank (X).*

$$
\begin{array}{|l}
X : \mathbb{N} \\
A, B : \mathbb{Z} \to \mathbb{Z} \\
\hline
X = 2, X \geq 0 \qquad\qquad \textit{[Account balance]} \\
A = B = \lambda\, x : \mathbb{Z} \bullet x - 1 \qquad\qquad \textit{[Requests]}
\end{array}
$$

The Bank enforces that the account X is unique at all times and enforces the strict access semantics. Hence processing concurrent withdrawal transactions from X requires that a valid sequentialization *exists. That is, an ordering of sequential transaction applications for which commonly understood semantics exist. Provided enough funds are present, the bank applies the two transactions to the bank account, so that only the final balance is left. In this case there are two different ways to sequentialize, and these are equivalent from the Bank's point of view.*

$$(A \fatsemi B)X = (B \fatsemi A)X = 0$$

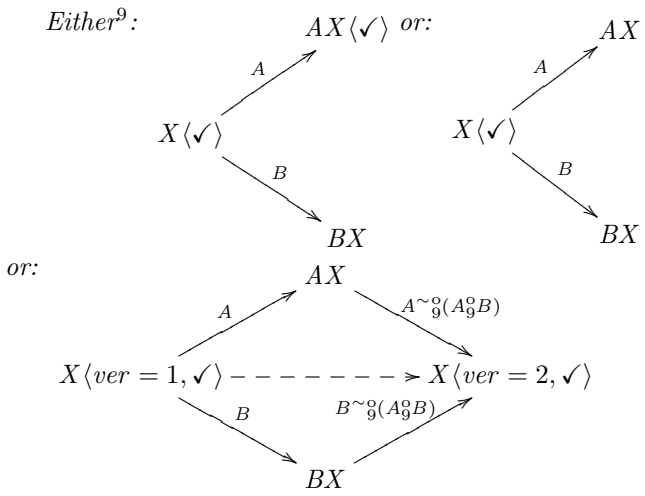
In the modified case, $X = 1$, only one of the requests can be fulfilled, as far as the Bank is concerned. This would annoy either Alice, or Bob, but not both as only one of the requests can be performed while keeping the account balance non-negative.

Now turn to the Distributed Blackboard (DB) case. Regardless of the state of the bank account, both requests can be executed on the account object, yielding a new state. All objects are simultaneously exported by the EMX of all the participants.

Question arises which of the produced objects is the "real" state of the project account after concurrent accesses have been performed. Only all *the objects considered at once encode the full state. But, what is "real" depends on the viewpoint. Likely, from Alice's point of view, AX being the real state is just enough. For what we know, the converse holds for Bob. And both would agree that it would be great for the original balance X to remain. What prevents these dreams from becoming true is the authority of the Bank. The authority is drawn from real finance that the Bank controls. A Bank signing some of the objects (denoted as: ✓) it means: "the Bank claims that this is the 'real' state".*

[7] *The concurrent access to a shared variable is considered a classical example and is described in many books. The blackboard bank account example is invented here, however.*

[8] *Such an account can have several researchers who independently and concurrently draw funds from it.*



This prevents neither Alice nor Bob to still privately hope otherwise: no-one prevents you to consider yourself rich when you are likely not[10]. But it does not do any good to either of them (as the vendors are likely to believe the Bank) and they might just as well accept the truth. As far as the storage for the objects is concerned, in this context there is no incentive for anyone to remember the old account state[11] X given that the authority has updated.

This illustrates the important difference in the approaches that use DB to those of the conventional transaction-based systems. On a DB, an object, once posted is never removed or modified. When it becomes necessary to represent a change to the object on the DB, it is done by producing a similar object with a different annotation and maintaining a relation between objects which denotes how the new object relates to the old. In the example, a version stamp (*ver*) and the bank signature (✓) are used. It is a matter of external convention to ensure that the objects on the DB fulfill the constraints placed on them. The DB clients are free to choose these conventions and even have several of them concurrently in effect.

4.6 Related Work

The motivation for the presented design is the introduction of automated decision support systems for the crisis management domain all the way through the

[9] *The blackboard* arrow diagrams *show possible blackboard contents at a given time. The arrows show the production dependence, and the arrow inscription shows the transform that has been applied to the source object to obtain the destination object.*

[10] The meaning of "being rich" is relative and depends on what one sees as relevant for being rich. We stay on the pragmatic track and not mandate that meaning. Instead we allow multiple meanings to exist concurrently, and it is up to each participant to choose the one which is relevant for the given context.

[11] The Bank may keep the records for own bookkeeping but these need not be exported.

management hierarchy to the individual operational units. There exists the need to support crisis management at the operational level, and the needed support is clearly identified in [81]: the presented system "[...is] usable by people who will have an understanding of their roles [and] will allow the individual users a high level of tailoring, filtering [and] will allow the operation of the response function without the need for a single operational physical center [and will be a] structured communication process independent of the nature of a particular crisis." However, the topic has involved comparatively little research given that service guarantees are difficult to make. This has been known since the consensus impossibility results ([52] and [51], Theorem 5.1).

The need to formalize computational tasks for crisis management led us to explore clean formal ways of task denotation. The decomposition STS item (i.e. unpacking tasks with hierarchical structure) is the inverse of *composition* (i.e. packing tasks into higher level tasks) of the NELSIS design methodology (see [83]), where it has been used for the (seemingly unrelated) VLSI[12] design. The derived task structure is naturally described by CPNs (see [69]), whereas the specifications for the task itself as well as the blackboard operation can be described effectively using the Z language (see [14], [28]). The task structuring led to regular and straightforward implementation in the COUGAAR framework (see[10]).

According to [81], several authors pointed out that "the critical general propert of computer-based [group] communications [is that] the content of the information being communicated [is] also the 'address' for delivery." This idea took various forms in the literature. Probably the earliest appearance is in Bloom filtering (see [11]), where hash-based discrimination is used. It appears as a generalization of the IP-numbering based addresses in [18]. The typical data space based approach is Linda and its various implementations. A complementary is the event notification (or publish-subscribe) approach, in which events are communicated and filtered, and are then removed from the system.

The system outlined in the introduction naturally compares itself to similar peer-to-peer (P2P) systems. The early P2P systems such as Napster [64], Gnutella [36], FreeNet [32] and MojoNation [62] were the first to introduce the promise of ever-present file storage, and were a motivation for this work. Subsequent P2P systems, such as Tapestry [92], Chord [63], Pastry [71], Kademlia [53], Content Addressable Network [68] discuss efficient key-based routing in various settings. These methods support deterministic message routing in an overlay network. These P2P overlays establish well-behaved distributed structures. However, they support only the Distributed Hash Table (DHT) interface, thus can only store key-value pairs of the form $\langle \text{key} \mapsto \text{value} \rangle$, where the key is distributed uniformly over a (large) key space. This key-based addressing is unfortunately not powerful enough for content-based addressing. This is not a problem as it takes DHTs out of their context. They do however set the path that task distribution structures as given here should follow.

[12] Very Large Scale Integration.

Practical distributed systems often use indirection of system services to achieve the flexibility of the task mapping. In Section 2.3 of [79], two mechanisms for indirection are mentioned. First, "the agent requesting or providing a service communicates with appropriate yellow pages server". This line of design yields systems that are vulnerable to yellow pages server failures, an unwanted side effect. The second approach where "the agent requesting or providing a service broadcasts this fact to all appropriate agents", admits the event notification and data space approaches. The usefulness of event notification in dynamic networks has been recognized (in [8]), and tree event distribution structures investigated (in [42]). A tuple-space implementation, albeit with disregard for efficient distribution is Lime (in [3]). We revisited the efficient distribution structures, and efficient activation in [59] to show a set of approximate matching techniques (employed in an unrelated context in digital logic design in [23]) that simplify the distribution.

4.7 Summary

In this chapter we addressed the *task mapping* for DWEAM, a process that transforms the application requirements for a distributed system into a specification that can be implemented and subsequently executed by the DWEAM system. The specification is not a direct part of the DWEAM computer architecture, but defines the DWEAM development environment. Given its fairly unconventional nature when compared to mainstream object-oriented systems, it is important to give its detailed description.

We start from the task descriptions that we expect to be useful in emergency rescue operations. From there the generic task descriptions are extracted and refined, recognizing the Static Task Structure with task description and task properties. We then formalized the task structure. We described, proved and demonstrated how the formalization fits into the executable CPN model, as a flexible executable structure admitting the task description. Thereafter, we describe the mapping of tasks to nodes, via the CPN.

Then we present the design constraints, an important by-product of the immersion of a computer system into the real world. We point out the impossibility results and give examples thereof, leading to probabilistic restatement of the problem at hand. We then present the Distributed Blackboard used for the task distribution, and introduce the DB semantics by which the only allowed operation on the content is the addition. A consequence of such an access semantics is that the objects on the DB must be allowed to exist in multiple versions, and that insisting on object uniqueness in the traditional sense does not apply. Rather, we conclude that it must be allowed that multiple, possibly even conflicting views exist. We compare this approach by example to traditional distributed systems, where an unique fixed semantics is enforced.

A number of issues are deliberately left open in the exposition. We give a short account of these issues here and explain how they are further dealt with.

First, we give no argument that the task-level descriptions are adequate to capture the crisis management tasks in full. Unfortunately, this is due to the fact that we have found the crisis management procedure descriptions to be rather informal, subject to positive regulations, and hence so far exclusively intended for human use. At the time of this writing, there is no formalized procedure that compiles these procedures into the structured task form that APL expects as input. The likely reason for the lacking description is that the use of information systems in crisis management is at its infancy and is still regarded as somewhat of an art rather than a codified discipline. This means that an adequate representation can be produced on demand, but that automated representation is still not feasible. The automated representation asks for independent research unrelated to our main topic, and would require prohibitive resources. Thus, instead of presenting the formalization we do the next best thing. We assume that such a description can *in principle* be formulated, and offer a minimal interface from that description to the task structure we presented. This design decision allowed us to proceed with the design of the DWEAM system.

Second, the properties of the DM as introduced in Section 4.5 are temporarily left open. It is easily seen that the publish semantics that does not allow object removal must eventually exhaust the available memory at some nodes. This important issue is resolved in subsequent chapters. We will show that, although the nodes are not able to control the memory they use for the object storage explicitly, a form of *garbage collection* will guarantee timely reclaim of the unused objects. In fact, we will show how the knowledge of the process structure, obtained through the data provided by the clients taking part in the DM helps the garbage collection. While the unbounded memory requirement is a valid criticism, there exist practical applications that work disregarding the unbounded memory requirements. Example applications are the garbage collection mechanisms for the Java language (which allows the allocated memory to grow without bounds), and the version control systems such as CVS or Subversion (where the records corresponding to versioned resources do not expire event if the resources themselves are removed). The authors of practical applications thus have some freedom to violate the rules of good conduct, when such violation can be justified. We too have used this freedom judiciously in our design of DWEAM.

Chapter 5

The Environment and Storage Model

The following Chapter describes the environment model assumed throughout the design of the DWEAM architecture. The environment model accounts for the node and network volatility effects. Due to these effects, the availability of computing resources (i.e. the nodes and the network) changes over the system lifetime. Knowledge about the environment which is distilled in the appropriate model is used to understand how the adverse influence of the environment causing volatility effects can be prevented. To this effect, in this Chapter, the basic properties of the environment model are presented.

5.1 Introduction

The DWEAM system consists of two distinct resource types: the *nodes* and the *network*. The nodes abstract a collection of computers (PDAs, laptops and such), typically carried by their users, while the computing facilities are being exploited. This usage pattern is termed *mobile user*, to signify that the user changes physical location while using a service provided by own local computer. It is important to make a distinction between the mobile and the *nomadic* user model. In the latter, the user only accesses the network infrastructure from various physical locations, but the mobility while the network is being used is negligible. This other usage scenario corresponds for instance with using the same laptop to access an Internet-based service both from home, and from work. The user carries the laptop from home to work and back, and uses it at both sites. However, the laptop is not being used in transit.

In the mobile user scenario each node has an attributed physical location. This location may change over time, as the user changes its position. The network resources at disposal in the mobile user scenario are typically radio-based. The prevalent use case for the radio-based communication nowadays is that based on

GPRS, for which the existing communication infrastructure is used, such as that of the mobile phone (e.g. GSM) networks. The GPRS usage scenario assumes that a fixed communication infrastructure exists, and is terminated by the *base stations*. Each base station is used as a portal to the infrastructure for the users that are geographically close to that base station. Each user communicates only via a base station to any other user on the communication network. This holds even for users that are at a given time using the same base station for access (and are likely geographically close by).

In the DWEAM architecture, we consider an extended communication scenario, that drops the requirement for the existence of base stations, and a fixed communication infrastructure. Instead, the users need to cooperate in message delivery. Each user originally has a very limited view of the entire network, and can directly contact only the nodes which are relatively geographically close. This is because of the *path loss* properties of the radio network. The network formed in this way is called a Mobile Ad-Hoc Network (MANET).

The MANET Properties

The MANET is induced by the time-variable graph of nodes and links, G_t. The set of active nodes ($active\_nodes(t)$) at a given event time t depends on the node availability that is induced by the environment. There typically exists an interval of event times $I == [t_1 : \text{Time}, t_2 : \text{Time})$ during which a given node $x \in Nodes$ appears in $active\_nodes(t)$, for some $t \in I$. thus any such node x comes into existence at t_1, and then lives on until it is permanently removed from $active\_nodes$ at t_2. Further, the existence of the connection in $active\_links(t)$ depends in part of the position of the nodes through time. For each node in $active\_nodes(t)$, its position in a d-dimensional space is given as a function of real time.

$$position : Nodes \times real\_time \nrightarrow \mathbb{R}^d$$

$$\forall (x : Nodes, \tau : real\_time) \bullet$$
$$(x, \tau) \in \operatorname{dom} position \Rightarrow \exists\, e \in Events \bullet$$
$$RealTime(e) < \tau \land x \in active\_nodes(e)$$

The goal of the MANET building is the establishment of a structure for communication between the network members. A lot of research is directed at understanding the fundamental MANET properties. An account of MANET research has been given in [4]:

> "Architectures have been primarily driven by a 'point-to-point' philosophy. We need to better understand a network viewpoint wherein nodes can cooperate intelligently taking advantage of the special properties inherent in wireless communication."

> "Architectures have been primarily centralized. We need to develop highly distributed architectures and algorithms that are still robust and energy-efficient on a system basis."
>
> [...]
>
> "Research efforts have been primarily compartmentalized. We need highly inter-disciplinary research across signal processing, communications, game theory, and networking."

To enable an integrated treatment of the MANET design issues, its relevant properties are given in the list below.

1. *Point-to-Point Connectivity.* This property regards the density of the direct connections between the nodes in the network.

2. *Connectedness and Structure.* This property regards whether the MANET provides connectivity so that any pair of nodes can connect either directly, or via multiple hops.

3. *Capacity and Throughput.* The capacity regards the maximum information exchange rate that can be achieved in a given MANET under the ideal transmission schedule policy. The throughput regards the information exchange rate that can be achieved under the adopted (likely suboptimal) transmission schedule policy.

4. *Service Discovery.* This property concerns the manner by which it is determined which nodes need to communicate with each other, and communication channels are established between them.

We now examine the MANET properties in more detail.

Point-to-Point Connectivity

The radio communication is based on the broadcasting of radio waves, at a certain power level. With the increase in the physical distance, the power level of the signal when it is received typically decays inversely proportionally to some power of the distance. For a given transmission power level ρ, and a geographical distance r between the receiver and the transmitter, the received power level p is:

$$p = \frac{\rho}{r^{\alpha}}, \tag{5.1}$$

where α is a path loss exponent. Typically, $\alpha = 2$, although α can take on other values depending on the propagation model. We are interested in the large scale effect here, so will not elaborate on the possible values of α. The Signal-to-Noise-and-Interference Ratio (SINR) has two principal components: the *ambient noise*, whose instantaneous power is assumed constant and denoted as N; and

interference comprising of the signal powers of all the transmissions that are active at a particular time instant. Given T, the set of all interfering nodes, X_t for $t \in T$ their positions, X_s the position of the sender and X the position of the receiver, the SINR is given by:

$$\text{SINR} = \frac{\rho \| X_s - X \|^{-\alpha}}{N + \rho \sum_{t \in T} \| X_t - X \|^{-\alpha}}. \tag{5.2}$$

The condition for the successful reception of a sent transmission from some node s is that the resulting SINR is greater than some threshold β. The exact value of β depends on the adopted coding and modulation schemes, but regardless of the actual value of β, the general condition for successful reception of the form:

$$E_{\text{success}} \equiv \text{SINR} > \beta$$

is valid for many communication scenarios. The probability of an error a transmission, the *outage probability* is the probability of the event E^c_{success}, a complement of E_{success} from the equation above. It is usually required that for successful communication it be bounded by prescribed level ϵ:

$$\Pr\left(E^c_{\text{success}}\right) \leq \epsilon. \tag{5.3}$$

Connectedness and Structure

Point-to-Point connectivity defines the conditions under which there can exist a direct connection between a sending and a receiving node. With the Point-to-Point as a basic element for connectivity, it is a logical followup to ask what is then the resulting connection structure in G_t. In particular, it is important to know under which conditions all nodes in G_t are connected, either by a direct link, or indirectly via a sequence of Point-to-Point links. The issue of connectedness has been studied extensively in the continuum percolation theory (see Meester and Roy [54]), from which our model for the connectivity is adopted. Here the percolation model for the connectivity concerns the structure of $position(x, \tau)$ for any node x and a given real time stamp τ.

We model the node positions by a *stationary point process* (see Meester and Roy, [54], page 9). That is, we assume that each node has a randomly chosen position in the space $\mathbb{R}^d$. The dimension d can be an arbitrary positive integer. Being application driven, we only consider the case $d = 2$, yielding a node configuration on a two-dimensional surface (e.g. plane). To make the notion of a point process more precise, one considers the set $\mathbb{R}^d$, and the σ-algebra of Borel sets in $\mathbb{R}^d$ as $\mathcal{B}^d$. Denote by N the set of all counting measures on $\mathcal{B}^d$ for which the measure of a point is at most 1. Equip N with a σ-algebra $\mathcal{N}$ generated by the sets of the form:

$$\{n \in N \bullet n(A) = k\}$$

where $A \in \mathcal{B}^d$, and k an integer.

A *point process* X is a measurable mapping from a probability space $(\Omega, \mathcal{F}, P)$ into $(N, \mathcal{N})$.

The definition of $\mathcal{N}$ allows one to count the number of points in a set $A \in \mathcal{B}^d$. In words, $X(A)$ represents a random number of points inside A. A point process X is *stationary*, if its distribution P is invariant under an arbitrary translation.

The point process X is said to be a Poisson process with density $\xi > 0$ if the following are satisfied:

1. For mutually disjoint sets A_1, A_2, ..., A_k the random variables $X(A_1) \dots X(A_k)$ are mutually independent.

2. For any bounded set $A \in \mathcal{B}^d$, we have for each $k \geq 0$:

$$\Pr\left(X(A) = k\right) = e^{-\xi \|A\|} \frac{\left[-\xi \|A\|\right]^k}{k!}, \tag{5.4}$$

where $\|A\|$ is the measure of the area A. The parameter ξ is the density of the corresponding Poisson process.

For the node connectivity, the description of a Poisson point process is ammended. Following Meester and Roy, a Poisson Random Connection Model (RCM) is set up. The RCM describes the connection structure in a d-dimensional space induced by a non-homogeneous Poisson process. Again $d = 2$ is assumed. For $A_i \subseteq \mathbb{R}^d$ ($d \in N$), a random process X is a non-homogeneous Poisson process with density $\xi > 0$ if the following properties are satisfied:

1. For mutually disjoint sets A_1, A_2, ..., A_k are mutually independent.

2. For any bounded set $A \in \mathcal{B}^d$, we have for each $k \geq 0$:

$$\Pr\left(X(A) = k\right) = e^{-\xi \int_{\mathbb{R}^d} g(x) \mathrm{d}x} \frac{\left[-\xi \int_{\mathbb{R}^d} g(x) \mathrm{d}x\right]^k}{k!}. \tag{5.5}$$

The function $g : \mathbb{R}^d \to [0, 1]$ is the *intensity function*. For our purposes, g is the distance-based connectivity between the nodes. We assume a static model in which the following holds, with r and q vectors in $\mathbb{R}^d$:

- $g(r) = g(q)$ whenever $\|r\| = \|q\|$,
- $g(r) \leq g(q)$ whenever $\|r\| \geq \|q\|$.

The derivation for $g(r)$ for the MANET is given further in the text.

Capacity and Throughput

MANETs are formed by mobile computing devices equipped with radio antennas that are used to communicate. Usually the range for the successful communication is limited, so communication between distant nodes is only possible if nodes cooperate by relaying messages to each other. In [39], Kumar and Gupta assumed an optimal routing scheme to discover the throughput capacity of an arbitrary MANET of n devices, with an available radio channel of capacity W bits per second, in an unit area is of the form $\Theta(W/\sqrt{n \log n})$. Their analysis is based on the following assumptions:

1. The traffic patterns in the MANET are random. Pairs of devices, one transmitter and one receiver, are chosen independently at random and packets are sent between them.

2. The packet routing is optimal. The packets are delivered along the shortest path between the transmitter and the receiver.

Under the transmission models given in [39], these two assumptions lead to the achievable throughput of an arbitrary MANET. The achievable throughput as derived in the said paper is, in fact, the *raw* throughput. It is unreasonable to assume that it reflects the useful throughput too. This is because the analysis is conditioned on the assumed known traffic pattern. The sender and receiver pair groups are assumed pre-determined by an unspecified device that has been abstracted away of the analysis for simplicity reasons. In real-life applications it is necessary to determine the traffic pattern before any of the wireless connections can be used. This is named *service discovery*.

5.2 Connectivity Function

In this section, we derive the analytical form of $g(r)$, the connectivity function of the RCM in the MANET case. Consider a receiver placed on a planar surface. Assume that its position determines the coordinate origin of an appropriate coordinate system. To derive $g(r)$ one needs to find the probability that, under the transmission model of the equation (5.3), a transmission from a transmitter at distance r from the origin is successfully received. The interference is due to the ambient noise, assumed to be time-invariant and with mean power equal to N, and the signals produced by the transmitters that operate independently but concurrently to the considered transmitter.

Exact Analysis

For successful transmission, for all the nodes transmitting simultaneously, equation (5.2) must hold. Consider a circle C with the radius R on the planar surface.

According to the RCM, the probability of an event:

$$\{X(C) = k \text{ nodes are present in } C\}$$

is given by:

$$\Pr(X = k) = e^{-\xi\|C\|}\frac{[\xi\|C\|]^k}{k!}, \tag{5.6}$$

with $\|C\| = R^2\pi$, the area of C. A transmission from a node at a distance d from the origin succeeds with the following probability:

$$g(d) = \mathrm{E}\left[\Pr\left(\frac{\rho d^{-\alpha}}{N + \rho\sum_{j=1}^{K}\|X_j\|^{-\alpha}} \geq \beta|K\right)\right]. \tag{5.7}$$

For the circle C, assuming independent uniform distribution of transmitters, the Probability Density Function (PDF) of D, the random variable denoting the distances from the receiver at the origin is given by:

$$f_D(r) = \begin{cases} \frac{2r}{R^2} & \text{for } r \in [0, R] \\ 0 & \text{otherwise.} \end{cases} \tag{5.8}$$

Rearranging the non-interference condition, one obtains:

$$\sum_{j=1}^{K}\|X_j\|^{-\alpha} \leq \frac{d^{-\alpha}}{\beta} - \frac{N}{\rho}. \tag{5.9}$$

From the equation (5.9) and non-negativity of $\|X_j\|$ for all j, it is verified that $g(d) = 0$ for d fulfilling:

$$d \geq \left(\frac{\rho}{\beta N}\right)^{1/\alpha}. \tag{5.10}$$

From equation (5.8), by the inverse theorem the PDF for the random variable $D^{-\alpha}$ is:

$$f_{D^{1/\alpha}}(x) = \begin{cases} \frac{2}{\alpha R^2 x^{2/\alpha+1}} & \text{for } x \in \left[\frac{1}{R^\alpha}, \infty\right) \\ 0 & \text{otherwise,} \end{cases} \tag{5.11}$$

and the corresponding generating function is given as:

$$G(s) = \int_R f_{D^{1/\alpha}}(x)e^{-sx}\mathrm{d}x, \tag{5.12}$$

which yields:

$$G(s) = \frac{2\Gamma(-2/\alpha, s/R^\alpha)}{\alpha R^2 s^{2/\alpha}}, \tag{5.13}$$

where $\Gamma(a, x)$ is the upper incomplete Gamma function:

$$\Gamma(a, x) = \int_a^\infty \mu^{x-1}\, e^{-\mu}\mathrm{d}\,\mu\,. \tag{5.14}$$

A sum of K random variables as in equation (5.9) has the generating function $G^K(s)$. The corresponding PDF is then given by:

$$f(x) = \frac{1}{2\pi i} \oint_M G^K(s) e^{sx} \mathrm{d}s. \tag{5.15}$$

The equation (5.15) is, however, unlikely to yield a useful result, as $G(s)$ cannot be further simplified. We leave the Equation (5.15) at that, and resort to finding an approximate solution to the problem.

Approximate Solution

Resorting to an approximate solution, assume that K is large with high probability. For large K, the equation (5.7) yields itself to Gaussian approximation. However, we will not approximate all the transmitters with a Gaussian distribution. Looking at Equation (5.1), it is seen that the interference from the far-away nodes must be low, as it decays fast with the distance. Also, the expected number of such nodes is high, as the expectation of Equation (5.4) is proportional to the size of the considered area. Hence for this area the Gaussian approximation is justified. The transmitters are therefore split into two groups: the *near* and the *far* group (Figure 5.1). The *near* group consists of the transmitters between

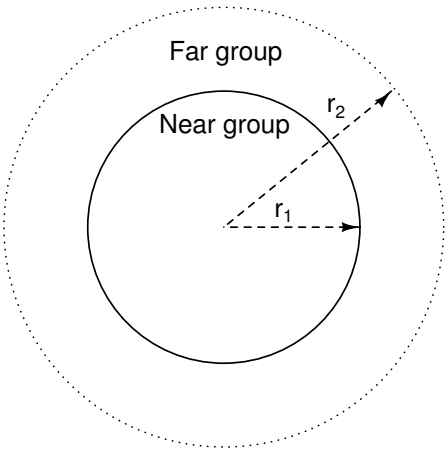


Figure 5.1: The near-far group approximation for computing the connectivity function $g(d)$.

0 and r_1. The *far* group consists of the transmitters between r_1 and r_2, where $0 < r_1 \leq r_2 \to \infty$.

The first limit, r_1 is the *critical radius* (also denoted further on as r_c), designating an area within which any single interfering transmitter completely prevents

the message communication with the receiver at the origin. The second limit, r_2 in principle can be set arbitrarily far away, and further in the text we assume $r_2 \to \infty$.

Theorem 2 (Critical radius) *Consider the planar arrangement of nodes, with a receiver placed at the coordinate origin. Also consider a transmitter, at distance d, with the path loss exponent $\alpha > 2$. The critical radius r_c is given as:*

$$r_c = \frac{d^{-\alpha}}{\beta} - \frac{N}{\rho} \tag{5.16}$$

Proof. Consider the non-interference model, as given in the Equation (5.2). Assume that there is only one interfering node, i.e. $\#T = 1$ at distance r from the receiver. Assume the receiver and the transmitter are at a distance d. Then:

$$\text{SINR} = \frac{\rho d^{-\alpha}}{N + \rho r^{-\alpha}}. \tag{5.17}$$

From the receive condition SINR $\geq \beta$, substituting the expression for SINR from Equation 5.17, and introducing renaming $[r_c/r]$, one obtains the claim. □

Considering that the critical radius r_c is non-negative, one can again obtain the limiting condition for d for any connectivity to exist. Observing that $r_c \geq 0$ and using Equation 5.16, one obtains the same condition as was given in Equation 5.10.

Function $g(d)$ is in effect the probability of the connection existence given the distance between the receiver and the transmitter. The following proposition is naturally derived from this assumption.

Proposition 1 (Disk Connectivity) *Let $g(d)$ be the probability of the connection existence. Then the connectivity in the sense of Equation (5.3) is:*

$$\Pr\left(E_{success}\right) = [g(d) > \epsilon] \tag{5.18}$$

where $[\cdot]$ is the Iverson (indicator) function [37].

From the proposition 1 it is seen that the area (with respect to d) has the shape of a disk, provided that $g(0) > \epsilon$, as g is monotonic and non-decreasing, as required by the model given in the Section 5.1. We will now give an expression for $g(d)$.

For this purpose, we divide the planar area outside of C into concentric, infinitely thin annuli. As in the Section 5.2 it can be seen that trying to account the exact contribution of all of the interferers leads to expressions which are impractical to evaluate (the connectivity expressed through the upper incomplete Gamma function), the only resort is to try a suitable approximation.

We attack this problem using the common balls-and-bins approach, as follows. Consider the annuli into which the $\mathbb{R}^2 \backslash C$ is divided as "bins", each carrying some number of "balls", each "ball" being an interferer node, which happens to be located at this particular bin. Let $X(r)$ be the interference contribution for the

annulus ("bin") $\mathrm{d}C$ at the distance r from the origin, and $\mathrm{d}r$ thick, with $\mathrm{d}r \to 0$. The probability $\mathrm{d}\Pr(n(\mathrm{d}C) = k)$ that there are k interferers in this area is:

$$\mathrm{d}\Pr(n(\mathrm{d}C) = k) = \frac{\partial}{\partial r}\left[e^{-\xi r^2 \pi}\frac{(\xi r^2\pi)^k}{k!}\right]\mathrm{d}r. \tag{5.19}$$

for $k \in \mathbb{N}_0$, and the contribution of each bin is then:

$$\mathrm{d}X(r)r^{-\alpha} \tag{5.20}$$

where $\mathrm{d}X(r)$ is a random variable with the distribution given in Equation (5.19). The total interference is then given as:

$$Y = \int_{r_c}^{\infty} r^{-\alpha}\mathrm{d}X(r). \tag{5.21}$$

While it is still involved to compute the distribution of the cumulative interference Y exactly, given that it is the sum of independent random variables, the Gaussian approximation may be invoked. For this, $\mathrm{E}[Y]$ and $\mathrm{Var}[Y]$ must be computed. The variable $Z = (Y - \mathrm{E}[Y])/\sqrt{\mathrm{Var}[Y]}$ is then the Gaussian-distributed unit variance, zero mean random variable. An elementary lemma helps to obtain $\mathrm{E}[Y]$ and $\mathrm{Var}[Y]$:

Lemma 1 *For a random variable $V = \sum_i V_i$, of independent random variables V_i, the following holds:*

$$\begin{aligned}\mathrm{E}[V] &= \sum_i \mathrm{E}[V_i] \\ \mathrm{Var}[V] &= \sum_i \mathrm{Var}[V_i].\end{aligned} \tag{5.22}$$

Using Lemma 1, the moments for Y are obtained.

Proposition 2 *Let Y be a random variable, defined as in Equation 5.21. Then, $\mathrm{E}[Y]$ and $\mathrm{Var}[Y]$ are given as:*

$$\begin{aligned}\mathrm{E}[Y] &= \frac{2\xi\pi}{\alpha - 2}r_c^{2-\alpha} \\ \mathrm{Var}[Y] &= \frac{4\xi^2\pi^2}{2\alpha - 3}r_c^{3-2\alpha} - (\mathrm{E}[Y])^2.\end{aligned} \tag{5.23}$$

where $\alpha > 2$.

Proof. By Equation 5.21, we have:

$$\mathrm{E}[Y] = \int_{r_c}^{\infty} 2\xi r\pi r^{-\alpha}\mathrm{d}r = \frac{2\xi\pi}{\alpha - 2}r_c^{2-\alpha}, \tag{5.24}$$

conditioned on $\alpha > 2$. Similarly, for $\mathrm{E}\left[Y^2\right]$, one obtains:

$$\mathrm{E}\left[Y^2\right] = \int_{r_c}^{\infty} 4\xi^2 r^2 \pi^2 r^{-2\alpha} \mathrm{d}r = \frac{4\xi^2\pi^2}{2\alpha - 3} r_c^{3-2\alpha} \tag{5.25}$$

conditioned $\alpha > 3/2$. Together with the condition of Equation 5.24, it amounts to $\alpha > 2$. □

By central limit theorem:

Proposition 3 *The random variable* $Z = \frac{Y - \mathrm{E}[Y]}{\sqrt{\mathrm{Var}[Y]}}$ *has Gaussian PDF with zero mean and unit variance, where* $\mathrm{E}[Y]$ *and* $\mathrm{Var}[Y]$ *are found in Proposition 2.*

The successful reception condition in Equation 5.7 (i.e. the conditional thereof) is rewritten in terms of Z to be:

$$g(d \mid n(C) = 0) = \Pr\left(Z \geq \frac{d^{-\alpha}/\beta - \rho/P - \mathrm{E}[Y]}{\sqrt{\mathrm{Var}[Y]}}\right), \tag{5.26}$$

which is then given in terms of the function $\Phi(z) = (2\pi)^{-1/2} \int_{-\infty}^{z} \exp(-z^2/2)\mathrm{d}z)$ to be:

$$g(d \mid n(C) = 0) = 1 - \Phi(\frac{d^{-\alpha}/\beta - \rho/P - \mathrm{E}[Y]}{\sqrt{\mathrm{Var}[Y]}}), \tag{5.27}$$

for d constrained in the sense of the Equation (5.10).

We have therefore proven the following:

Theorem 3 *The connectivity function* $g(d)$ *is given by:*

$$g(d) = \left[1 - \Phi(\frac{d^{-\alpha}/\beta - \rho/P - \mathrm{E}[Y]}{\sqrt{\mathrm{Var}[Y]}})\right] e^{-\xi r_c^2 \pi}, \tag{5.28}$$

with all the quantities appearing in Equation 5.28 as defined previously in the text.

Thus, starting from the basic assumption about the distribution of the nodes, stemming from the RCM model, we derived the approximate condition for the function $g(d)$, denoting the probability of successful reception (hence: the existence of a connection) between a transmitter and a receiver at a distance d, in the presence of an unbounded number of interferers, distributed uniformly at random in $\mathbb{R}^2$.

The range function depends on $g(d)$ as: $[g(d) > \epsilon]$, where ϵ is a parameter depending on the techniques used to modulate the transmitted signal. As $g(d)$ is monotonically non-increasing, the shape of the successful reception area is always a disk. However, independent of the modulation technique, there is a critical radius r_c, for which all transmission necessarily stops. This critical radius is due to the existence of the "background" noise that slowly but surely drowns the signal, as the distance between the transmitter and the receiver increases. In this

Chapter, both the upper bound $\hat{g}(d)$ and the Gaussian approximate for $g(d)$ are computed in terms of the system parameters.

The implication for the connectivity structure is strong. The connectivity $g(d)$ imposes a limit to the number of nodes that can be contacted by any given node. Moreover, the nodes are within a limited radius. This implies that a data structure that requires "far reaching" connections can not be implemented with constant resources per node. This because for a node to reach a distant receiver it must use a number of intermediate nodes proportional to the distance between it and the destination, divided by at least the maximum range. A node must consider the soft state recorded by all the intermediate nodes, and this cannot ever be made to be significantly less than the stretch of the graph.

5.3 The Storage Model

In this section, we turn the attention to the reliability of storing data to nodes in a volatile network. We present an analysis of using the fragmentation of data into *tags*, and depositing the tags onto various nodes in a multi-agent network, to achieve resilience against node volatility. We emphasize Multi-Agent Systems (MASs) as example platforms in which there is system-wide interest in preserving data. Agent programs in a MAS often use an *agent platform* as middleware [50]. The platform controls the messaging and the agent life-cycle. The reliability of the agent platform has received little attention *in practice*, though it is recognized as a MAS issue [80].

Notable MASs, such as JADE [85] or COUGAAR [10], assume a reliable platform. Sometimes the platforms provide persistence to non-volatile media, protecting the agents from transient failures. For both platforms, replication services exist; these services have been implemented with simplicity in mind, rather than efficiency. We use information-theoretic arguments to show how efficient reliability in a MAS can be derived from cooperation with a slight trade-off for increased complexity. The resulting platform is called Combined [1]; this platform is based on the COUGAAR agent architecture, with additions that make it suitable for deployment in chaotic, rapidly changing environments.

Application

The main motivation for looking at increasing the reliability of multi-agent platforms is because it is desirable in applications. We will mention two applications which benefit from the Combined approach.

Emergency search-and-rescue operations benefit from timely information delivery. This deployment setting was also adopted as the prototype application for Combined. We envision a near-future application in which members of emergency service units (police, fire brigade, ambulance etc.) are equipped with short range communication devices. Due to the nature of the deployment, it cannot assume

that communication infrastructure exists; the reliability of the communication devices can also not be taken for granted. The devices work around these impediments by cooperating in message delivery and keeping each other's observations; the body of the Chapter explains the theory on which this possibility is based.

Large scale distributed scientific applications, such as Seti@Home [74], or Folding@Home [31] succeeded in harnessing worldwide processing power for highly demanding computational tasks. The focus of their system architectures has shifted from traditional parallel and distributed computing efficiency issues [51] to organizing and managing redundant work done by worldwide computers. Some indication of the system architecture can be found at [13]. The Combined approach might aid this effort by offering a way to preserve computation results despite computers leaving the network.

Related Work

The system outlined in the introduction naturally compares itself to similar peer-to-peer (P2P) systems. The early P2P systems such as Napster [64], Gnutella [36], FreeNet [32] and MojoNation [62] were the first to introduce the promise of ever-present file storage, and were a motivation for this work. Subsequent P2P systems, such as Tapestry [92], Chord [63], Pastry [71], Kademlia [53], Content Addressable Network [68] discuss efficient key-based routing in various settings. These methods support deterministic message routing in an overlay network. OceanStore [46] is an architecture for establishing global persistent storage; similarly, the Cooperative Filesystem (CFS) is an application that uses Chord to make a P2P-based read only storage system. Unlike Combined, all these systems work with an overlay on top of an existing Internet Protocol (IP). The underlying IP network layer allows cheap contact between any pair of the participants. The interconnect mesh corresponds to a complete graph. However, Combined is established on a spatial proximity based network mesh, with connections induced by the distance between nodes. Due to this external constraint, the interconnect mesh is much sparser than that of the mentioned systems. The contact between far away nodes can be costly. Typically, the transitive closure of the Combined interconnect mesh would correspond to the IP-based interconnect used by the mentioned P2P systems. Another important distinctions is that in Combined, cooperation is needed for even the simplest operations such as contacting a far away node. In the mentioned P2P systems, this operation is handled by a lower level network protocol. The layering simplifies the design, but limits the applicability of the overlay to networks in which IP is efficient. The ideas of Combined are most thoroughly shared with OceanStore. This architecture uses erasure coding to achieve high data reliability. The improvement from using erasure coding over replication was clearly shown in another paper [87] from the co-authors of OceanStore. OceanStore considers erasure coding only for deep archival; but also goes to considerable length to handle data security. Combined intends to use data archival to provide process persistence; but it does not consider data security explicitly. This difference is

easily understood in the light of the Combined prototype application. In such a setting, it is reasonable to assume that cooperative behaviour by far outweighs competitive or malevolent behaviour.

Result

We present a graph-theoretic model using *erasure graph* to describe the system interconnect and evolution, and *partition encoding* for allocating data pieces (tags) to different agents in Section 5.4. We then use this model to derive the storage capacity of the resulting society, and prove that the capacity cannot be larger than the availability figure of the most reliable agent in the network (Section 5.4). Assuming the availabilities are known, we describe the strategy that an agent can use to determine which code and data partition to choose to make best use of the available capacity (Section 5.4). We first give a calculation for a general interconnection case, and random agent position within the network. We then simplify the analysis by layering the neighbours in tiers and allowing agents to delegate pieces of work to each other.

5.4 System Model

We now give an overview of the system setup and notation. For detailed definitions and more flexible models, the reader is referred to [56]. The network of nodes is constructed based on the interconnections created by a short range radio network: two nodes are connected if they are closer to each other than a pre-determined *range*. This is a reasonable assumption, shared by papers closely dealing with wireless network properties, such as [39]. Denote as V the set of nodes $\{v_1, \ldots, v_m\}$. If a connection between two different nodes v_i and v_j exists, we insert the pair $\{v_i, v_j\}$ into the set E. The graph $G(V, E)$ is interpreted in the usual sense: V is a set of vertices, and E is a set of corresponding edges. When we need to emphasize that V or E belong to a graph G, we write V_G and E_G, respectively. The interconnection mesh of G changes as the nodes move, join or leave the network. When nodes leave the network, it can happen either *voluntarily* or *involuntarily*; the latter can occur because of network partitions or nodes being damaged, out of power, or destroyed. The involuntary leave is called an *erasure*. Due to nodes leaving the network, the interconnection graph *degrades*. If a node v leaves, it is removed from V, along with the corresponding edges from E. Here we assume that each node can leave independently with identical probability ε. We call the graph G the *erasure graph*.

Consider a *source node* v from V that has during its lifetime produced valuable data (*a tag*) that should be preserved in case that v gets erased. Node v can choose to deposit copies of the tag with its neighbours. Depositing copies is commonly called *replication*. This approach, although effective, uses too much excess storage. Moving tags also spends bandwidth and a smart source node would want to minimize the spending. In [87], an alternative approach was discussed, whereby

the node uses an erasure code to transform the tag and deposits only fragments of the resulting coded messages to neighbours. Thanks to the coding, only a fraction of the fragments is required for successful decoding. There exist efficient methods (for instance, the iterative decoding method from [70]) to recover original tags. OceanStore and CFS [22] use these methods, but assume that the missing tags were erased independently, and those that remain are accessible independently. In the locally-connected network mesh, these assumptions do not necessarily hold. When trying to deposit a tag, v will have a choice of contracting several neighbours and request them to be tag *keepers*. Typically, v will want to pre-code the tag as said before copying to the keepers. v would then make a *partition* of the entire message. The partition is described by a set $\Pi = \{\pi_1, \ldots, \pi_m\}$. Each element of the partition is in itself a set denoting which bit of the encoded message is allocated to which node. For example, if the message was 5 bits long, and 2 nodes were available, one possible partition would be: $\Pi = \{\pi_1, \pi_2\}$, where $\pi_1 = \{1, 2\}$ and $\pi_2 = \{3, 4, 5\}$. This means that bits 1 and 2 are allocated to the node with index 1, and bits 3, 4 and 5 are allocated to the node with index 2.

To store the fragments, the source node v chooses the code and the partition. It then contacts the nodes, as defined by the partition, and deposits the message fragments. This is called a *write*. To retrieve the stored information, v contacts all the available keepers and retrieves the fragments. The fragments are then re-assembled and decoded to produce the original tag. This is called a *read*. Between the writes and corresponding reads, the keepers may be erased, or may be unavailable at the time of the read. Although each keeper has a probability ε to disappear, its availability also depends on its connection to other nodes. We will denote the availability of the node v_i from V as w_i. The writer will typically want to know in advance the availability of the potential keepers to decide which partition is the "best". Due to erasures, the graph G could be partitioned into connected components. We denote as $\mathcal{Q}_G$ the set of all maximal connected components for a given graph G. These maximal connected components are called *chains*. Two simple writer decisions are given in the following examples. The decisions are named according to the cooperation policy: the *Lone Ranger* prefers to keep all the message to itself; the *Cloner* prefers to make multiple identical copies of the message.

Example 9 (The Lone Ranger) *The source decides to keep the entire message on a single node. The probability that the message is readable is:* $\Pr(\textit{readable}) = \Pr(\textit{keeper alive}) \cdot \Pr(\textit{path to keeper exists}) \leq \Pr(\textit{keeper alive}) = \varepsilon$.

While the total storage is minimized, any transition that erases this single keeper destroys the tag.

Example 10 (The Cloner) *The source node decides to duplicate the tag with k keepers. This ensures that if at least one keeper is present, the entire tag can be retrieved. However, the case in which such storage space investment is justified is very improbable. The probability that all but one keeper are absent is:* $k(1-\varepsilon)\varepsilon^{k-1} \leq k\varepsilon^{k-1}$, *and tends exponentially fast to zero with k.*

While the data cannot be protected better, the frequency of the catastrophic event is too low for the storage spending to be justified.

The Capacity of the Erasure Graph

Let there be given a connected graph G. For the given graph G, a writer would like to know the storage capacity that G can offer. This section provides tools to do just that. Let us denote as X the message that a writer needs to distribute across the network formed by nodes V. For convenience, we consider that X is a string of n binary digits (bits). As in practice all data objects used in a computer system can be represented in this way[1], we only need to consider the binary strings further in the text.

Each bit of the message X should be stored to some node in G. As the nodes in G are volatile, it is to be expected that not all the bits from X can be recovered once they are handed out to nodes in G. From that perspective, G behaves as a lossy erasure channel: the writer stores data expressed as bit-strings into it and the reader retrieves a damaged bit string, with some bits irreversibly lost due to the volatility of the nodes that stored them. The reader will be able to retrieve only some fragments of the original message (call the fragments Y), due to the degradation of the network. We will consider each bit of X as a *channel use*

The mutual information[2] on X and Y is given as: $I(X,Y) = H(X) - H(X|Y)$, where $H(X)$ and $H(X|Y)$ are, in order, the entropy of a random variable X and the conditional entropy of a random variable X given Y [21].

Since when X is known, then any Y is known, it holds that: $I(X,Y) = H(Y) - H(Y|X) = H(Y)$, i.e., knowing X gives all the information over Y. The writer has some freedom to choose the partition Π, if it knows the availabilities (written as: w_i) for the nodes v_i from V.

This way, $H(X|Y)$ becomes a function of Π. Knowing all w_i, writer can choose the Π that maximizes $H(X|Y)$. The capacity of G is then $C = max_{\Pi} I(X,Y)$. We now express C in terms of w_i in Theorem 4, and find C explicitly in Theorem 5.

Theorem 4 (The Capacity of G given Π) *Let G be an erasure graph. The capacity of the channel defined on the graph G, under a given partition encoding Π and assuming uniform connection probability, is obtained by solving a linear program:*

$$C = max_{\Pi} \sum_{1 \le i \le \# V} w_i \# \pi_i; \quad \sum_{1 \le i \le \# V} \# \pi_i = n, \tag{5.29}$$

[1]Encoding objects for transfer into binary strings explicitly is common practice. Facilities to achieve this are present in many programming languages, and libraries where the feature is not provided by the language.

[2]The meaning of mutual information in this case is simply the number of bits shared by X and Y. Likewise, the entropy $H(X)$ is the number of bits in X, and the conditional entropy $H(X|Y)$ is the number of bits that are left unknown in X if we know all the bits of Y.

where coefficients w_i for $1 \leq i \leq \#V$ depend on the connectivity of the graph G:

$$w_i = \sum_{e=0}^{\#V-1} \frac{\varepsilon^e (1-\varepsilon)^{\#V-e}}{\#V - e} \cdot \sum_{E_p \in \mathcal{P}_e} \sum_{H \in \mathcal{Q}_{E_p \circ G}} \#V_H \left[v_i \in V_H \right]. \tag{5.30}$$

In Equation (5.30), E_p is the *erasure pattern*, a function mapping each vertex of G into the set $\{0, 1\}$ and associated with a particular transition. It maps $v \in V$ to 1 if v is erased by the transition, or to 0 if v is not erased by the transition. E_p can be *applied* to $\mathcal{Q}_G$, to obtain a new connected components set: each vertex that E_p maps to 1 is removed from G along with corresponding edges. $\mathcal{Q}_{E_p \circ G}$ are the connected components that G is split into, after applying the erasure pattern E_p. $\mathcal{P}_e$ is the set of all erasure patterns on G, having exactly e erasures. A consequence of Theorem 4 is the maximum capacity obtainable for a given graph G.

Proof. The probability of e erasures on a single transition is given by:

$$\Pr\left(e \text{ erasures}\right) = \binom{\#V}{e} \varepsilon^e (1-\varepsilon)^{\#V-e}.$$

Generate the set of all possible erasure patterns for a given e and name it $\mathcal{P}_e$. Every $E_p \in \mathcal{P}_e$ induces a set of connected components $\mathcal{Q}_{E_p \circ G}$, and each of the sets is composed by connected graphs $G_{Ei} \in \mathcal{Q}_{E_p \circ G}$. The mutual information $I\left(X, Y\right)$ is the mean of the number of bits retrieved from all the possible erasure patterns. The set of all possible erasure patterns $\mathcal{P}$ is given by the union of all individual erasure patterns: $\mathcal{P} = \bigcup_{0 \leq e \leq \#V} \mathcal{P}_e$. The average number of bits retrievable from the channel depends on the particular transition, and ultimately of the erasure pattern within. Considering e known and looking at a particular erasure pattern E_p, the average number of bits is obtained by averaging over all possible connection points, since by assumption of the theorem, a reader can be with uniform probability connected to either of the remaining nodes of $\mathcal{G}$.

$$I\left(X, Y | \mathcal{Q}_{E_p \circ G}\right) = \frac{\sum_{G \in \mathcal{Q}_{E_p \circ G}} \#V_G I\left(X, V_G\right)}{\sum_{G \in \mathcal{Q}_{E_p \circ G}} \#V_G}. \tag{5.31}$$

From here it is easy to obtain that:

$$I\left(X, Y\right) = \mathrm{E}\left[I\left(X, Y | \mathcal{Q}_{E_p \circ G}\right)\right] \tag{5.32}$$

will yield the expression for mutual information for X and Y. These expressions depend upon elements of Π. The expression in Equation (5.32) includes multi-dimensional sums, which ultimately depend on the (beforehand unknown)

connectivity of graph G. Let e remain fixed, and let us focus on Equation (5.31). For a given $v \in V$, define $\langle v \rangle$ to be the index of v. Substituting $I(X, V_G)$ by:

$$\sum_{v \in V_G} \#\pi_{\langle v \rangle},$$

it is seen that $\#\pi_{\langle v \rangle}$ enters the sum on a number of occasions.

It is possible to determine how many times and with what weight coefficient does $\#\pi_{\langle v \rangle}$ appear in the appropriate equation, and the answer depends on e and on the size and number of the connected components that v can belong to. Let the connected components be named *chains*, for convenience. The shortest chain that v can be part of has size 1, when v is its only element. The longest chain has size $\#V$, when no degradation takes place. If v belongs to a chain of size l, when e errors are present, its contribution to expression (5.32) depends on the position of v in the string.

There are different ways in which v can be a member of a chain of size l. Expanding the expectation in Equation (5.32), one obtains

$$\begin{aligned} I(X, Y) &= \mathrm{E}\left[I\left(X, Y | \mathcal{Q}_{E_p \circ G}\right)\right] \\ &= \sum_{E_p \in \mathcal{P}} \frac{\sum_{G \in \mathcal{Q}_{E_p \circ G}} \#V_G I(X, V_G)}{\sum_{G \in \mathcal{Q}_{E_p \circ G}} \#V_G} \Pr\left(\mathcal{Q}_{E_p \circ G}\right), \end{aligned} \tag{5.33}$$

and averaging the number of bits retrieved from each element of $\mathcal{P}$ will give $I(X, Y)$. Since the degradation model is i.i.d., the probability of each $E_p \in \mathcal{P}_e$ is equal to $\varepsilon^e (1-\varepsilon)^{\#V-e}$. By introducing an indicator function it is possible to restate $I(X, V_G)$ as:

$$\begin{aligned} I(X, V_G) &= \sum_{v \in V_G} I(X, v) \\ &= \sum_{i=1}^{\#V} I(X, v_i)\left[v_i \in V_G\right] \\ &= \sum_{i=1}^{\#V} \#\pi_i \left[v_i \in V_G\right], \end{aligned} \tag{5.34}$$

permitting the separation of the sum in Equation (5.31) as:

$$\begin{aligned} I(X, Y) = \sum_{E_p \in \mathcal{P}} &\frac{\Pr\left(\mathcal{Q}_{E_p \circ G}\right)}{\sum_{G \in \mathcal{Q}_{E_p \circ G}} \#V_G} \cdot \\ &\cdot \sum_{G \in \mathcal{Q}_{E_p \circ G}} \#V_G \sum_{i=1}^{\#V} \#\pi_i \left[v_i \in V_G\right]. \end{aligned} \tag{5.35}$$

Substituting $\mathcal{P}$, Equation (5.35) becomes:

$$I(X, Y) = \sum_{E_p \in \bigcup_{0 \le e \le \#V} \mathcal{P}_e} \frac{\varepsilon^e (1-\varepsilon)^{\#V-e}}{\sum_{G \in \mathcal{Q}_{E_p \circ G}} \#V_G} \cdot \cdot \sum_{G \in \mathcal{Q}_{E_p \circ G}} \#V_G \sum_{i=1}^{\#V} \#\pi_i \left[v_i \in V_G \right]. \tag{5.36}$$

When E_p is an element of $\mathcal{P}_e$, then the following expression holds:

$$\sum_{G \in \mathcal{Q}_{E_p \circ G}} \#V_G = \#V - e, \tag{5.37}$$

which is easily seen to be true, as $\sum_{G \in \mathcal{Q}_{E_p \circ G}} \#V_G$ is the total number of nodes present in the connected component $\mathcal{Q}_{E_p \circ G}$.

By changing the order of summation in Equation (5.33) such that the first sum goes over all π_i, and noting that for $e = \#V$ the sum component is equal to zero, one is able to find: $I(X, Y) = \sum_{i=1}^{\#V} w_i \#\pi_i$, where w_i is given by Equation (5.30). □

Theorem 5 (The Capacity of G) *Let the channel be defined on the graph G, and let all availabilities w_i be known for all nodes v_i from V. Let:*

$$w^* = max_{1 \le i \le \#V} w_i.$$

Then: $C = nw^*$.

This is to say that, regardless of the choice of the partition Π, it is not possible to obtain the rate (ratio C/n) greater than the maximal availability of all the nodes in G.

Proof. Let w_i and π_i be permuted, without loss of generality, so that $w^* = w_1 \ge w_2 \ge \cdots \ge w_{\#V}$.

$$C = max_{\Pi} \sum_{1 \le i \le \#V} w_i \#\pi_i \le w_1 \sum_{1 \le i \le \#V} \#\pi_i = w_1 n, \tag{5.38}$$

since for each i, $w_i \#\pi_i \le w_1 \#\pi_1$. □

The Choice of the Partition

We have seen how the node availabilities affect the storage capacity achievable by a given graph G. We will now describe the strategy the writer uses to find the "best" partition. In [56], the subset of acceptable partitions is captured by the notion of distortion in the following definition.

Definition 8 (Distortion) *Consider some erasure graph G. Let $\mathcal{Q}_G$ be the set of connected components of G, and let the partition Π be known for distributing a message X over nodes in V of G. The* distortion *of some $\mathcal{Q}_G$ (written as: $d(\mathcal{Q}_G)$) equals to the expected tally of irretrievable message bits of X, over all possible erasure patterns.*

Thus, the distortion tells us for the connected components $\mathcal{Q}_G$ of some graph G, how many bits are expected to be be lost, when the graph undergoes an arbitrary erasure pattern.

Proposition 4 (Distortion Properties) *Consider the distortion as given in the Definition 8. Consider some transition between the components $\mathcal{Q}_G$ and $\mathcal{Q}_{E_p \circ G}$. Denote as D_c be the set of all possible connected component configurations obtainable by applying an erasure pattern E_p to $\mathcal{Q}_G$, and let $V(\mathcal{Q}_{E_p \circ G})$ be the set of all vertices thereof.*

1. *Let $R = (1 - \varepsilon_T)n - \sum_{i \in V(\mathcal{Q}_{E_p \circ G})} \#\pi_i$. Then the* distortion. *Then:*

$$d(\mathcal{Q}_G) = \sum_{\mathcal{Q}_{E_p \circ G} \in D_c} \Pr(E_p) \cdot R \cdot [R > 0]. \tag{5.39}$$

2. *Let π_i be the partitions for nodes v_i. $d(\mathcal{Q}_G)$ is a non-negative piecewise linear function of $\#\pi_i$.*

Proof. The property 1 is immediate. For property 2, note that every term of Equation 5.39 is non-negative. All $\#\pi_i$ are non-negative. Setting all π_i to be empty sets (corresponding to maximum distortion), the maximum of $d(\mathcal{Q}_G)$ is given by:

$$d(\mathcal{Q}_G) = (1 - \varepsilon_T) \sum_{\mathcal{Q}_{E_p \circ G} \in D_c} \Pr(E_p) \cdot [(1 - \varepsilon_T)n > 0] = (1 - \varepsilon_T)n. \tag{5.40}$$

By re-expressing $d(\mathcal{Q}_G)$ in terms of $\#\pi_i$, we get:

$$d(\mathcal{Q}_G) = a_{0,\mathcal{C}} - \sum_i a_{i,\mathcal{C}} \#\pi_i, \tag{5.41}$$

where $a_{0,\mathcal{C}} = (1 - \varepsilon_T)n$, and $a_{i,\mathcal{C}} \neq 0$ in general. □

Given a maximum distortion $d_{\max}$ and the area

$$v(d_{\max}) = \{\mathcal{Q}_{E_p \circ G} \mid d(\mathcal{Q}_{E_p \circ G}) \leq d_{\max}\},$$

the best configuration $\mathcal{Q}_G^*$ is given by

$$\mathcal{Q}_G^* = \arg min_{\mathcal{Q} \in v(d_{\max})} f(\mathcal{Q}), \tag{5.42}$$

where f is a disambiguation function that helps in the choice of the unique solution. The parameter ε_T is the *decoding threshold* [70] of the employed decoder, the fraction of n that can be erased, without entailing decoding error. f is a goal function picked according to design criteria:

1. Given a threshold ε_T, decoding must succeed if erasure fraction is less;

2. Given two partitions from $v(d_{\max})$, the encoder chooses a "more distributed" one;

3. The encoder must handle a variable number of hosts ($\#V$), and variable message lengths n.

The requirement 1 is implicitly satisfied by choosing a threshold-ε_T code. Further requirements can be satisfied in different ways. We choose the partition Π such that the sum-squared of all fragment lengths ($n_i = \#\pi_i$) is minimal. The choice amounts to using:

$$f(\mathcal{C}) = \left(\sum_i (\#\pi_i)^2 \right)^{1/2}, \tag{5.43}$$

effectively choosing as the configuration to be at the the point closest to the all-zero configuration, i.e. the one where all bits on all nodes were lost.

This choice is akin to mean-square energy minimization in multi-channel signal detection: we assume that each fragment length contributes independently to the entire tag. The expected number of retrieved bits must be equal to $(1-\varepsilon_T)n$. This expression is precisely equal to the sum of n_i weighed by w_i for each node. Finally, the fragments should form a partition of the message, thus the sum of n_i must be equal to n. For the same reason, for all i, the condition $n_i \geq 0$ must hold. The distortion corresponds to the distance (in the number of lost bits) between

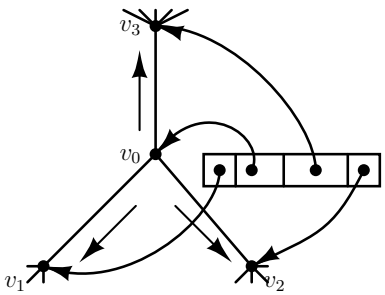


Figure 5.2: Subdividing V into subsets headed by nearest neighbours. v_0 is the writer. It delegates 3 fragments of a single tag to its nearest neighbours, v_1, v_2, and v_3. v_0 also delivers ζ_1 and ζ_2 to each of the neighbours so that they would be able to further sub-divide their fragment.

the complete message X and any received message Y. An example contour plot of $d(\mathcal{Q}_G)$ for a two-node configuration (labeled as 1 and 2) distortion is given in the Figure 5.3.

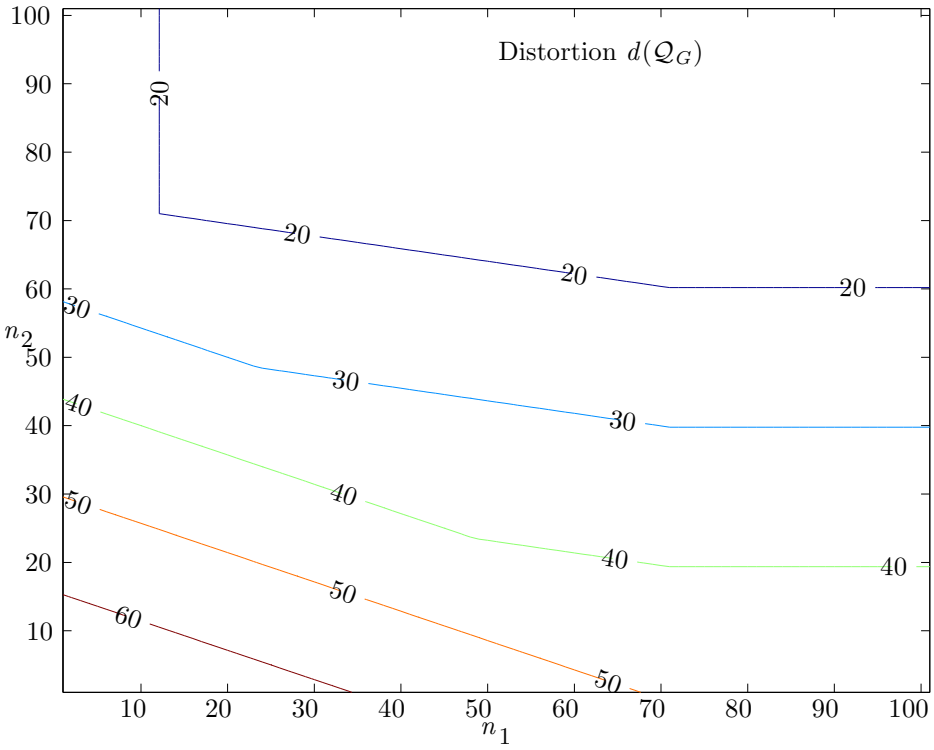


Figure 5.3: The level lines of a two-node configuration distortion, $d(\mathcal{Q}_G)$. The nodes are labeled 1 and 2, and number of bits stored in each are n_1 and n_2 respectively. The availabilities are $w_1 = 0.7$ and $w_2 = 0.3$. The message is 100 bits long.

5.5 Summary

This Chapter presented an information-theoretic approach to improving multiagent platform reliability. We show how, by using coding and partitioning, it is possible to achieve reliable data storage even when the platform itself is unreliable. We define a criterion for a feasible code that is used to choose the coding and partitioning. We show how the writers can then make first partitions, and then delegate their neighbours with sub-partitions. These information-theoretic arguments come from the vast information and coding theory literature but have so far received comparatively little attention in multiagent platforms despite potential gains. Multi-agent platforms that employ this (or similar) coding types can be made reliable enough to use in adverse, chaotic environments.

The exposition in this text assumes that the availability estimates are known

for all the nodes. In practice the estimates may be costly to obtain. It may be acceptable that a lower bound on the availability estimates is substituted. These can be obtained at comparatively little cost, by sub-dividing the nodes based upon the proximity to the writer, and then considering only a subset of possible ways that a node can be accessible to the writer. Apart from yielding the availability estimates, the sub-division can give rise to a distributed control algorithm, whereby once determined, the partition is allowed to dynamically change to compensate for changes in the network connectivity. This algorithm and its properties are recommended for further work.

Chapter 6

Core Based Tree (CBT)

6.1 Introduction

A method for a fully distributed solution to the SDP for a MASs under localized radio-based communication is described here. Informally, the SDP is the task of finding, between a set of *producers* and *consumers* of services those which are *compatible*, i.e. have an overlapping set of *product* (for the producers) and *consumption* (for the consumers) summaries.

The MASs considered here are established by connecting computational resources with a commercial, off-the-shelf wireless network. It is assumed for simplicity that each such resource (hereafter a *node*) hosts only a single software module (an *agent*), that computes a *partial result*[1] of a computational task. The partial result can be communicated to agents at other nodes. The agents must exchange the partial results in order to complete the entire task. The network used to connect the nodes is established by radio-based network devices (e.g. conforming to IEEE 802.11). Due to the interference and the path loss[2] of radio networks, any given node is only able to communicate with the few nodes that are geographically close to it. The only way to contact the nodes which are not reachable directly is by *hopping* through a sequence of intermediate nodes, where each adjacent pair of nodes are directly reachable. Each path taken between a pair of such nodes is called a *hop*. For a given node in the network, the number of the close by nodes (*neighbours*) is typically far less than the number of all nodes reachable through multiple hops. If the nodes can change their positions over time, a *mobile* network is obtained. Such a network is usually called a MANET.

[1] A partial result is an object constituting a part of a distributed computation. A distributed computation is an interleaving of the productions of partial results by nodes and the appropriate delivery of the partial results between nodes.

[2] The property of radio signals that their intensity decreases with the distance to the source. As the signal intensity decreases, the Signal-to-Noise Ratio (SNR) worsens, thus making successful signal detection more difficult with the increase in the distance from the source. The result is that radio-based connections exibit strong locality.

The MANETs have little initial structure. Typically, the only information a node has from the MANET itself is the set of its neighbours. It is assumed here that a node in the MANET maintains a current view of the neighbour set by some sensing method. Knowing the neighbours is of course not enough to guarantee the meaningful communication of partial results between agents. This is because:

1. The agents that need to communicate can be at a distance greater than one network hop so they cannot directly confer. This is due to the MANET properties and the decaying power of the radio transmission with distance.

2. The agents that need to communicate must determine which partial results should be communicated to which agent, knowing only the *summaries*, i.e. rough descriptions of the partial results that are being communicated through the network. This is because the network addresses, which encode the node's position within a network, have a short life-span due to the changes in the MANET connectivity.

The latter issue, called the Service Discovery Problem (SDP), is in the focus of this Chapter. Further in the Chapter the special properties of the SDP in the MANET environment are described.

Service Discovery

Service discovery is the name for a collection of techniques by which it is determined at runtime which nodes of a distributed system need to communicate. The service discovery precedes any actual data exchange. There usually exist sets of nodes that produce data objects (*producers*) and another node that consumes the same data object (*consumers*). The goal of service discovery is to convert the specification of the data object (i.e. a *service description*) provided by the consumer into a handle (i.e. *identifier*) by which the producer of the data can be contacted. Service discovery is one of the *resolution mechanisms* in routine used in the Transport Control Protocol (TCP)-based networks. The Domain Name Service (DNS) is an example of such a mechanism.

Example 11 (DNS) *The DNS, i.e. the naming service used on the Internet is a rudimentary service discovery mechanism. It converts the information need, in form of an URL of an object into an Internet Protocol (IP) address that can be used to contact the object provider.*

Every time before a consumer accesses an URL (such as: www.google.com*), it must address a DNS instance with an inquiry for an IP address. In this particular case (see Figure 6.1), a set of equivalent IPs is returned, with auxiliary data some consumers may find useful. After having obtained the IP number, the consumer can use it to address the producer directly by its network address.*

Thus, before actual communication occurs, an additional round of inquiries must take place, in which a consumer determines which producer can tend to its

```
; <<>> DiG 9.3.1 <<>> www.google.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 30373
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 6, ADDITIONAL: 6

;; QUESTION SECTION:
;www.google.com.                        IN      A

;; ANSWER SECTION:
www.google.com.         620     IN      CNAME   www.l.google.com.
www.l.google.com.       156     IN      A       66.102.9.99
www.l.google.com.       156     IN      A       66.102.9.104
www.l.google.com.       156     IN      A       66.102.9.147

;; AUTHORITY SECTION:
l.google.com.           63598   IN      NS      g.l.google.com.
l.google.com.           63598   IN      NS      a.l.google.com.
l.google.com.           63598   IN      NS      b.l.google.com.
l.google.com.           63598   IN      NS      c.l.google.com.
l.google.com.           63598   IN      NS      d.l.google.com.
l.google.com.           63598   IN      NS      e.l.google.com.

;; ADDITIONAL SECTION:
a.l.google.com.         63875   IN      A       216.239.53.9
b.l.google.com.         63875   IN      A       64.233.179.9
c.l.google.com.         63875   IN      A       64.233.161.9
d.l.google.com.         63875   IN      A       64.233.183.9
e.l.google.com.         63875   IN      A       66.102.11.9
g.l.google.com.         63875   IN      A       64.233.167.9

;; Query time: 5 msec
;; SERVER: 130.161.180.1#53(130.161.180.1)
;; WHEN: Thu Feb  2 15:58:32 2006
;; MSG SIZE  rcvd: 292
```

Figure 6.1: The standard response of a DNS server, when queried for a server named *www.google.com*.

information need. The service that generalizes service discovery is named Yellow Page Service (YPS). The YPS translates service descriptions (infomation needs) into Unique Identifiers (UIDs) of the receivers that offer the service matching this need.

We describe in brief two typical methods for building the YPSs, the *Centralized YPS* and *Broadcast YPS*. We then describe in brief an improved schema that we call the *Minimally Distributed YPS*, which relies on the environment model as built in this text.

Centralized YPS

A fixed subset of nodes is chosen to keep and update the service descriptions. The centralized YPS is the solution with the simplest structure of all. As the YPS delivers UIDs, representing the network-layer address of the receiver, stale UID often means that by the time YPS is finished resolving, it is impossible to refer to the receiver as its position in the network is changed. This problem is pronounced in MANETs, where nodes can move about freely. Even when YPS is possible, it is still required for each client to have a network address of the YPS to be able to ask it for service.

Thus, introducing an indirection via YPS solves the information need based matching, but does not solve the tight coupling for the YPSs themselves. Moreover, the YPS node becomes a single point of failure. The outage of the YPS server means that its data must be rebuilt again elsewhere.

Broadcast YPS

Each node advertises its services to a subset of nodes that are closest with respect to some distance measure (such as network distance). A network-wide policy on further dissemination of service advertisements is adopted, by which is determined how the service descriptions are propagated.

The simplest distribution policy is that where all the service descriptions are propagated to all the nodes in the network. While this policy removes the single point of failure, it introduces the problem of the communication overhead and housekeeping of all the service descriptions across the entire network.

Minimally Distributed YPS

The minimally distributed YPS uses the smallest possible number of links for the service advertisements. This requires that a spanning tree structure is constructed and maintained throughout the network lifetime. For this purpose, the construction and maintenance of a spanning-tree strucure named CBT is presented.

Use Cases

The MANET scenario arises often, in practice and in the scientific studies alike. Diverse examples are given, that either use MANET or a network with similar properties for the communication. In these examples no fixed communication infrastructure is assumed to exist.

Sensor Networks. The sensor networks are formed by small measurement devices. These devices are equipped with sensors that measure various physical quantities (e.g. temperature, pressure, humidity), and a radio transceiver that can communicate and receive the measurements. The devices are "liberally sprinkled" in a target area. The MANET is used for communication between the nodes. The SDP in sensor network is trivial, and consists of forwarding the data from all the sensors to a single collection point.

Systems-on-Chip (SoC). The SoC are complex integrated circuits that combine a possibly large number of diverse circuits on a single silicon die. In such a system, it has been determined that simple bus-type connections yield performance bottlenecks and the networking approach is suggested. There is considerable interest in the research community for building a local interconnect for SoCs. The interconnect has MANET locality properties, although the physical data transfer uses metal layers rather than radio. The SDP for SoC consists of finding data paths.

Rescue Operations. The use of information technology in rescue operations has received increased interest in recent years. The participants in the near future rescue operations may be equipped with portable computers (e.g. PDAs) that will be used for critical data exchange, and to coordinate action. Such devices will expand the role of radio stations (e.g. *walkie-talkie*) from exchanging voice messages to full integration with the supporting information systems. It is reasonable to expect that such PDAs will be able to exchange data through radio links between themselves. These devices need not rely on the existing communication infrastructure. Rather, the devices need to be able to set up a MANET and cooperate in message delivery. The SDP for rescue operations consists of finding clusters of PDAs that need to communicate.

Contribution

The core contribution of this Chapter is in the new use of the structure known as the Core Based Tree (CBT) for providing the dynamic service discovery structure. Further, the detailed account of the CBT construction and maintenance algorithms, in the relaxed requirements context (i.e. allowing the appearance of unicycles) is novel to the best of our knowledge. Finally, using the Z and CPN frameworks to describe particular aspects of the algorithms (Z for describing the state changes, and CPN for handling concurrency) is novel to a limited extent. Since the state evolution of systems and concurrentcy handling is a recurring topic in the analysis of algorithms, there exist equivalent formalisms, which have

been used in various contexts: for instance, CSP in [40], CPN in [69], and CSP and Object-Z in [76].

The choice for CPN and Z stems from the straightforward way the presented algorithms are implemented in the used development platform, the COUGAAR framework [10].

Related Work

Papers describing related work can be roughly divided into three classes:

1. Papers that discuss *service discovery* in MANETs;
2. Papers about *event notification* (i.e. delivery of "event" objects to interested parties in a distributed setup); and
3. Papers about CBT in the context of *Internet multicasting*.

As reported in [65], the service discovery in MANET is organized as being either *service coordinator based* where designated nodes collect the service summaries (e.g. Jini [55], or virtual backbone approach of Kozat and Tassiulas [45]); or *distributed query based*, where service discovery requests are flooded through the MANET with respect to some *flooding scope* parameter (e.g. the *electrostatic field based* approach of [49], or efficient flooding of [67]); or *hybrid* (e.g. [73]), combining the former two approaches.

Our approach to SDP differs sharply from those given above as it avoids service coordinators as well as query flooding, reasons following.

First, the existence of service coordinators does not really solve the SDP for MANET, since the real issue is finding an efficient way to distribute the service descriptions, rather than the coordinator election. Even when coordinators are present, the problem of connecting them remains and is of the same type as the original SDP, although fewer nodes participate.

Second, query flooding as a way to compensate for not knowing the target position has scalability problems. However, it is the only way out as in MANET it is not known where the service queries need to go. Instead of flooding, a minimal structure is maintained that guarantees the participation of all the MANET nodes, and define a policy for forwarding the service descriptions over it. To achieve this, it is required that nodes in the CBT provide the descriptions of both the offered, and the required services.

The event notification has been extensively studied in the context of publish-subscribe middleware. Examples thereof are Gryphon [38] and Siena [17]. Contrary to our approach, these event notification systems are intended for use in a fixed network and hence do not need to construct a structure for service description propagation. However, they contain useful ideas which served as starting points in our work. These are the notions of service summaries and approximate service matching, introduced in [17], and analyzed independently in [91]. Applications of the efficient summaries under reconfiguration of the distribution structure

(analogous to topology changes in a MANET that is studied) was treated in [66] which are also considered feasible for MANETs.

The CBT structure is described in [9]. In that paper it is used as a way of space-efficient internet multicasting. Instead of building a multicast tree for each multicast session, the seminal CBT paper proposes using a single tree for all the sessions, thus simplifying the build up of the multicast infrastructure, at the expense of some extra incurred traffic.

The CBT idea is used in the modified context (i.e. MANETs) to make a distributed structure akin to a spanning tree, that is then used to guarantee successful service discovery. Making and maintaining the CBT turns out to be simple enough to be performed efficiently in a MANET. It represents a distributed spanning tree construction with the requirements somewhat relaxed compared to the original spanning tree problem formulation (Gallager et al. [34]).

6.2 Problem Description

Preliminaries

The basic type considered is called *Node*. It represents any system node. It is introduced as an opaque type as its internal structure does not play a role in the specification.

$$[\, Node \,]$$

The resource and network setup are described in terms of a family of graphs Γ, each member consisting of nodes and the connections:

$$\Gamma : \mathbb{R} \rightarrow \mathbb{P}\, Node \times (Node \leftrightarrow Node)$$

Each member of the family Γ is a graph represented by a pair $(V : \mathbb{P}\, Node, E : Node \leftrightarrow Node)$. Each element of V stands for a node and (with a tolerable abuse of notation) the co-located agent. E is a symmetric binary relation on V, and represents the nodes that can directly communicate between each other (hence the connections are always bi-directional). The family members of Γ are indexed by *time*. For a real t, $\Gamma(t)$ is the graph corresponding to the available nodes and connections at time t. Hence, Γ describes the temporal evolution of the network. Note that only a part of Γ can be retrieved at a given time t. This part consists of all $\Gamma(t')$ such that $t' \leq t$. It is assumed that each node can detect other nodes that it can directly communicate to. Call this the *neighbours set*.

$$N : Node \rightarrow (\mathbb{R} \rightarrow \mathbb{P}\, Node)$$

Given a node $v : Node$, and some time point $t : \mathbb{R}$, then $N_v(t)$ is the set of neighbours of v at time t. An obvious connection exists between N and Γ, such that $n : Node$ is only allowed to be a neighbour of v at time t if for $\Gamma(t) = (V, E)$ it holds that $(n, v) \in E$.

Each agent specifies the type of partial results that it accepts from other agents. Likewise it specifies the type of partial results that it produces for the other agents. The partial results considered here are drawn from a set of all constructible types, the dataspace:

$$[\ \mathit{Dataspace}\ ]$$

Hence all partial results are of the form $x \in \mathit{Dataspace}$. A node gives a description of all the partial results through predicates. Two predicate classes are used per agent. One detailing the acceptable elements of the dataspace, the other detailing the producible elements of the dataspace.

$$\phi^c, \phi^p : \mathit{Node} \nrightarrow \mathbb{P}\, \mathit{Dataspace}$$

Call the former the *consumer summary*, and call the latter the *producer summary*. For an agent v they are denoted as ϕ^c_v and ϕ^p_v respectively. For simplicity it is assumed that they do not change. The element sets of the *intersection* and *union* of two summaries ϕ_1 and ϕ_2, denoted as $\phi_1 \cdot \phi_2$, and $\phi_1 + \phi_2$ respectively define the sets:

$$\begin{aligned} \phi_1 \cdot \phi_2 &= \phi_1 \cap \phi_2 \\ \phi_1 + \phi_2 &= \phi_1 \cup \phi_2 \end{aligned}$$

A producer summary ϕ^p_v of a node v and a consumer summary ϕ^c_w of a node w are *compatible* if $\phi_v \cdot \phi_w$ is a nonempty set.

Problem Formulation

Now the SDP can be defined:

Definition 9 (Service Discovery Problem) *Let $t \in \mathbb{R}$ be a point in time. Let there be given a set of nodes and a family Γ as described. Let the families ϕ^p and ϕ^c be given. For each pair of nodes v and w such that ϕ^p_v and ϕ^c_w are compatible, find a path in $\Gamma(t)$ connecting v and w. Also notify v that w has been found, and vice-versa.*

By Definition 9, the SDP is a task continuously solved during the MANET runtime. The solution of the SDP allows any agent v to query whether there exists an agent w with a compatible service description. Once it is found, v can address w to render the advertised service. In a MAS an agent typically needs the rendition of the service rather than the contact with a particular agent by name, hence the need for this *lookup by service*.

Relevance

Lookup by service is typically [85, 10] realized in MAS as some form of a database, centralized for simplicity. This is a drawback in the MANET settings as reported

in [49]: "Service discovery in [MANET] is challenging because of the absence of any central intelligence in the network. Traditional solutions as used in the Internet are hence not well suited for [MANET]."

The SDP is an important component in the functioning of any MAS platform. It is a link between the distributed application running on a MAS, where agents connect based on the compatibility of their service summaries, and the network layer which is typically able to communicate messages between nodes identified by name. In a MAS that relies on a fixed node and network infrastructure, SDP can be solved to an acceptable level by database lookups [10]. In MANETs where the fixed node and network infrastructure assumption must be removed, the way to solve SDP becomes less obvious, although still crucial for the MAS operation. This importance of having an efficient solution to SDP for the MAS deployed on MANETs, without central coordination, is the main motivation for the work described in this Chapter, and it is viewed as an enabler for a wide range of MANET-based distributed applications.

The SDP as given in the Definition 9 is a compact formulation of a problem that arises frequently in MASs. A typical MAS consists of agents that both export *services* to the community and require services from other agents to work. In this context, a service consists of delivering some objects from *Dataspace* from the producer agent to the consumer agent. A consumer agent x that requires a service s can contract any produced agent y which can provide s. The consumer x does not care about which y is allocated to service s, as long as the service is rendered. It is said that x performs a *lookup by service*. The need for the lookup by service implies that a mapping must exist between service descriptions and the agents that offer this service. By analogy with telephone directories, such mapping is called the YPS (and is itself offered as a service). One approach of making the YPS is providing a registry that contains all the services and has an up to date mapping to the agents that implement them. The simplest form of the YPS registry uses a centralized database to store the service descriptions. For a large MAS a centralized database is a performance bottleneck and implementations consider distributed YPS that ameliorate the issue. Depending on the way the distribution is performed, the YPS servers may require some network traffic to synchronize their local registries. It is also required that the registry locations are known in advance. The database approach has a weakness when the underlying network is a dynamic MANET. In a dynamic environment it is not obvious where the YPS is to be hosted. The YPS locations are also out of reach for external configuration.

6.3 Solution Outline

The solution is proposed of the SDP by using a fully distributed data structure based on the Core Based Tree (CBT), and a detailed account is given how the structure is made and maintained. Before the solution analysis, its informal out-

line is given to explain what is achieved in the detailed algorithm description. The approach is in view of several *requirements* and *assumptions* as follows. Requirements for the data structure:

1. *Scalability.* Must operate efficiently with respect to the number of nodes;
2. *Simplicity.* Must be able to include new nodes in the structure by considering locally available information only; and
3. *Repairability:* Must be easily repairable in face of changes in network connectivity.

Assumptions:

1. *Slowly changing environment.* The CBT maintenance never terminates in the classical sense, constantly adapting as the connectivity changes, with the CBT property only partially fulfilled during adaptation. Frequent changes increase the probability that the CBT algorithm needs to adapt the CBT. Hence dynamics must be limited for the CBT properties to be fulfilled; and
2. *Infrequent change in service summaries.* Hence the change due to service description forwarding is localized only to parts of the CBT. This facilitates the CBT repair.

The CBT is used to provide a minimal communication structure offering full connectivity. The CBT is built implicitly: the nodes that form the CBT keep only a small amount of soft state that is used for local decision making. The CBT does not need to be stored entirely at any one node. The minimality stems from the scalability requirement, as the resulting structure must be easy to bookkeep, and require low bandwidth to manage. A tree-like structure follows naturally from these requirements.

The CBT is considered only as a structure for solving the SDP. The method is unsuitable for communicating the content as the traffic volume over the structure is not balanced. Thus using the CBT is proposed to solve the SDP, thereafter using any routing method appropriate for ad-hoc networks for content communication.

Core Based Tree (CBT)

For the rest of the exposition, Γ is assumed to be a family of connected graphs. Otherwise, assuming that for some t the $\Gamma(t)$ is disconnected, the given solution can be simply applied to all the connected components in turn. The solution is then valid within these components only.

The CBT is a weakly-connected[3] directed spanning tree over the set of nodes V that induce the graph $\Gamma(t) = (V, E)$ for some t (see the Figure 6.2 for illustration;

[3]A graph is weakly-connected if it is directed and there exists a pair of nodes u and v for which an oriented path exists either from u to v or v to u but not both.

the formal definition of a CBT and its invariants is relegated until further in the Chapter). A CBT is *maximal* in V if it includes all the nodes in V. It is also possible to have a forest of non-maximal CBTs that exaust V but as will be shown later, provided that $\Gamma(t)$ is connected, they can be joined to form a maximal CBT.

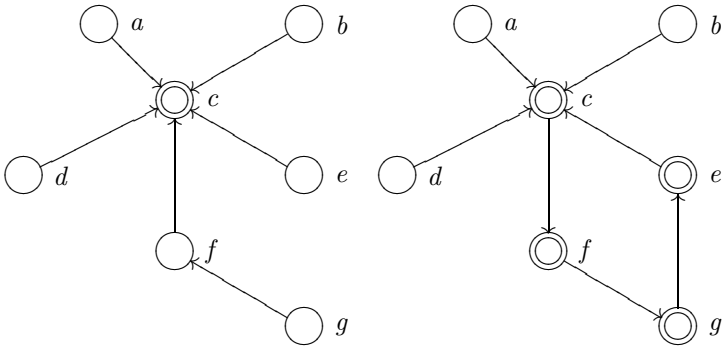


Figure 6.2: A CBT (left) and an unicycle (right).

Moreover, each node from V has at most one outgoing edge (henceforth: an *uplink*). The maximum outdegree of the CBT is one. It will be shown (see Lemma 3) that if these conditions hold there always exists an unique node whose outdegree is zero. This node is called the *core*. Such a CBT has always $\#E = \#V - 1$ edges, and checking that this condition holds verifies whether a structure is a CBT or not. Call this the *edge count* condition. It is also straightforward to prove that, when the MANET is connected, the edge count condition ensures that the underlying bidirectional graph is a tree.

The edge count condition entails much bookkeeping to discover and maintain at runtime, as the only way to check whether it holds is to execute a distributed edge count. But this is difficult to do when the connections are time-variable, as concurrent change in the connectivity invalidates an ongoing edge count. To simplify matters, the edge count condition is relaxed. Instead of requiring that the CBT has $\#V - 1$ edges at all times, it is required that it is connected and that each node in V has at most one uplink. It is easy to show that the class of admissible graphs increases only slightly and that it now includes, beside all CBTs, also the graphs that contain exactly one cycle. Call these *unicycle* graphs. It turns out that admitting unicycles to CBT simplifies the CBT formation, that it does not prevent the solution to SDP and that it is easy to repair. Figure 6.2 illustrates the CBT and the unicycle. Nodes are represented as circles, and the uplinks are arrows extending between pairs of nodes. On the CBT, the node emphasized with double frame is the unique core node c. On the unicycle, the emphasized nodes form the unique cycle (c, f, g, e).

CBT Construction and Maintenance

There are two main classes of events at runtime, where the CBT structure needs to be adjusted.

1. *On initialization.* Initially, the nodes of V are partitioned into V non-maximal CBTs, with only a single node each. A sequence of *join* operations is performed on the non-maximal CBTs until a maximal CBT is obtained.

2. *On change.* Whenever the connectivity of $\Gamma(t)$ changes and as a consequence the CBT conditions are violated, the resulting structure must be repaired. This happens when the connectivity structure in $\Gamma(t')$ changes with respect to $\Gamma(t)$ for some $t' > t$.

The CBT instantiation boils down to adopting an uplink for all the nodes in V, such that the CBT obtained at the end is maximal. This process exibits good structure, in the sense that the maximal CBT can be obtained by repeatedly joining smaller CBT. Hence our approach is to find and join in a distributed manner pairs of non-maximal CBTs. The *join* operation thus needs to perform the following:

1. Find two CBTs (say G and H) such that there exist edges in $\Gamma(t)$ that start at a vertex of G and end at a vertex of H (or vice-versa).

2. Denote the set of such edges as E_{GH}. From E_{GH} adopt exactly one edge (say e_{GH}) to connect G and H.

3. Adjust the edges in the union of G and H with e_{GH} added, and the position of the core nodes such that the obtained graph is a CBT again.

The join is divided into four distinct phases. Each phase represents a *transaction* that partially adjusts the CBT. As usual, the operations within a transaction are applied atomically to the CBT when a transaction is *committed*. Conversely, the operations within a transaction are not applied at all if the transaction is *aborted*. Within the bounds of a single transaction, the participating nodes are allowed to temporarily break the CBT invariants where appropriate. However, upon transaction commit, the resulting structure must fulfill the CBT invariants. The join unfolds in several phases. In Figure 6.3, the phases are named *Meet*, *Vote*, *Switch* and *Yield* and are described below. The dotted line on each illustration represents the CBT boundary. At *Meet* (top), two nodes belonging to different CBTs come into contact and exchange identity information. At *Vote* the same two nodes conduct a vote to determine which of the two CBT cores must yield its function. If the core of the yielding CBT is away from the node, a series of *Switches* are applied until the core node appears at the join. At *Yield*, the core node yields its function by adopting an uplink that leads to another component.

1. *Meet.* This phase is initiated between every pair of nodes that come to contact. It is used to set the communication up, and determine which node

is the leader for the join. The meet phase is used to prevent *crosstalk* (cf. [69], page 45), i.e. a situation in which both nodes either accept, or refuse to be the leader *ad infinitum*.

2. *Vote.* In this phase, the nodes take a vote to determine which of the cores of the two CBT has to yield. It is also used to determine whether a join is feasible. If yes, the control is passed on to the next phase. If no (such as when the nodes belong to the same CBT), the join is cancelled.

3. *Switch.* The node that executes a yield must be the core node. In case that, after a vote, the yielding node is not also the core, a sequence of Switch operations is initiated. The goal of each switch is to move the core node from its position to a node. A sequence of such switches is used to move the core of the yielding component to the join site.

4. *Yield.* Now that the core node is at the join site, it is enough that it adopts an uplink towards the far end of the join site. This implicitly removes the role of the core from that node. It can be shown that the resulting graph is a CBT, as required.

Cycle Removal

Several concurrent join transactions may unfold at any given time. As a consequence, parallel joins can occur. In Figure 6.4, the cores of the two components have each decided to yield (left). The yield directions are given by the dotted arrows. As a result, the obtained structure after the join (right) is an unicycle instead of a CBT. While parallel joins do affect the CBT structure (in fact, they destroy it), the resulting graph still provides full connectivity (hence supporting the solution to SDP), and is easily repaired, by removing a single extra link. For efficiency reasons the link removal is relegated to the point where the presence of a core node in a component is essential, i.e. to immediately before the Switch phase. This means that a graph remains an unicycle until it is required to join with another.

For the cycle detection to work, it is enough that the cycle is detected at any single node which forms a part of the unique cycle. How this is done will become obvious in the detailed discussion of the cycle removal. Once a node (say v) obtains a proof that a cycle exists it can take action to remove the cycle. For instance it may remove the cycle by dropping its current uplink. The node thus becomes the new core and may resume the usual operations. However, this approach has problems when concurrency comes into play. Multiple nodes may discover that they are a part of the cycle. If all of them are allowed to assume the role of the core, the graph would become unnecessarily fragmented, as more links would be removed than the minimum of one. This is not the best possibility although it is not catastrophic, as each of the fragments would itself form a CBT.

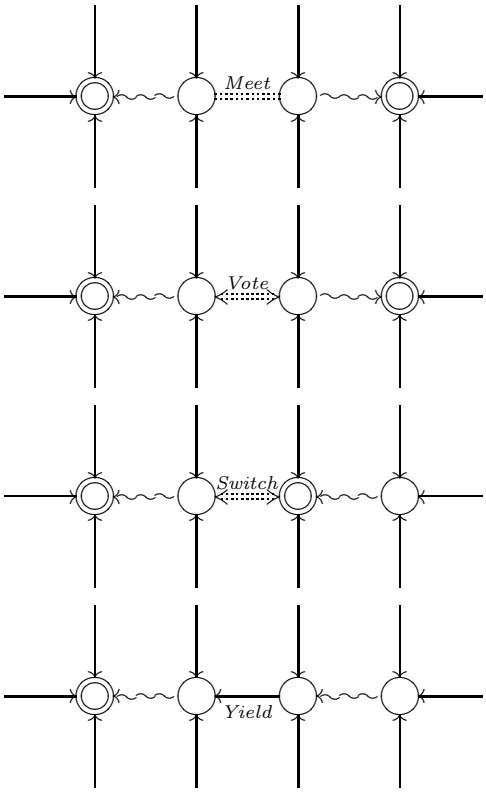


Figure 6.3: The phases of the *Join* operation.

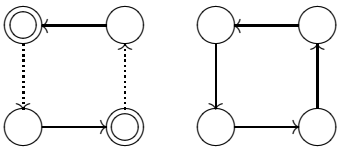


Figure 6.4: A parallel join.

An improved approach is to run a leader election algorithm in a ring. Leader election in a ring has a known solution ([51], page 475 gives several solutions). It is known from v that the ring (i.e. the cycle) is implicitly available by following a sequence of uplinks from the node that detected it. The node v can then initiate a leader election algorithm which is known to terminate eventually either with a success (whereby the elected leader removes its uplink and becomes the core), or with a failure (whereby the leader election is prevented by the cycle destruction;

but the cycle destruction is precisely what is wanted in the first place, so this outcome also wins).

SDP Strategy Computation

In this section it is assumed that at any given t, the CBT has been constructed on top of $\Gamma(t)$. Hence, for each node, the uplink is known if it exists.

$$
\begin{array}{|l}
upl : Node \times \mathbb{R} \pfun Node \\
coreof : Node \times \mathbb{R} \pfun Node \\
\hline
\forall\, t : \mathbb{R} \exists_1\, v : Node \bullet (v, t) \notin \mathrm{dom}(upl) \\
\forall\, t : \mathbb{R} \bullet u = coreof(n, t) \Rightarrow \exists\, n : \mathbb{N} \bullet u \in upl(\_, t)^n \land u \notin \mathrm{dom}\, upl(\_, t)
\end{array}
$$

Conversely, for a given node the set of downlinks is also defined. That is the set of the nodes that have it as an uplink.

$$
\begin{array}{|l}
dnl : Node \times \mathbb{R} \pfun \mathbb{P}\, Node \\
\hline
\forall\, t : \mathbb{R}, v : Node, W : \mathbb{P}\, Node \bullet \\
\qquad (v, t) \mapsto W \in upl \Rightarrow \forall\, w \in W \bullet dnl(w, t) = v
\end{array}
$$

For simplicity the changes in the CBT due to changes in $\Gamma(t)$ are not considered. Thus the time indices in *upl* and *dnl* are dropped for some fixed time point t. Assume that the nodes have the families ϕ^p and ϕ^c well defined. As a shorthand ϕ^x will be used, where $x \in \{p, c\}$ ranges over both summary types. As the CBT is hierarchical it is possible to aggregate the summaries as follows. The summary distribution unfolds in two interleaved phases. The first phase is the *convergence*, consisting of messages which get sent towards the CBT core. The second phase is the *divergence*, consisting of messages flowing from the core to the leaf nodes.

In the convergence phase, the leaf nodes forward the two summaries to the corresponding uplink. The non-leaf nodes collect summaries from their downlinks and make an aggregate with their own respective summaries by first computing an union and thereafter simplifying the resulting summaries where appropriate. The result of these operations are forwarded on to the CBT core.

In the divergence phase, a node examines the received summaries for compatibility. Compatible summaries have a non-empty intersection. Consider a node v for which two different nodes are downlinks as follows: $w, u \in dnl(v), w \neq u$. If $\phi^p_w \phi^c_u \neq \varnothing$, then v produces $\phi^p_w \phi^c_u$ and forward each to w and u. Similar matching occurs for v and w to handle the case when v and w are compatible.

6.4 Algorithm Description

In this section, the CBT formation and the summary matching algorithms are described in detail.

Preliminaries

Here it is assumed that the graph $G = (V, E)$ is a member of the family Γ, i.e. there exists some $t : \mathbb{R}$ such that $G = \Gamma(t)$. The label T is used when it is important that the graph has a special property (i.e. be a tree, or CBT, or unicycle). The standard terms for graphs are used throughout (see for instance [25]). First it is specified what is meant by CBT and Unicycle. Thereafter a connection is given between the two, also permitting the maintenance of a relatively weak set of invariants for the CBT construction, provided that it is known how to remedy any unwanted side effects when the CBT degenerates into an unicycle. The following Proposition specifies the invariants of a CBT.

Proposition 5 (Core Based Tree Invariant) *Let there be given a digraph*[4] $T = (U, H)$*, with* U *the set of nodes, and* H *the set of edges on* U*.* T *is a Core Based Tree (CBT) if the following hold in* T*:*

1. *The underlying*[5] *graph is a tree.*
2. *Any node* $u \in U$ *has at most one* uplink *in* T*.*

Definition 10 (Unicycle) *A* unicycle *is a weakly-connected digraph that has exactly one cycle.*

Lemma 2 (Directed Acyclic Graph (DAG) Endpoints) *Any DAG has at least one* source *node and at least one* sink *node.*

Proof. Without loss of generality, consider only the sink nodes. Let the number of nodes in the DAG be n and suppose the contrary, that no node is a sink. Then for each node, there is at least one outgoing edge. Follow an outgoing edge to a new node. Repeat the procedure $n + 1$-times to obtain a path of length $n + 1$. As there are n nodes, by Dirichlet principle there is a node that has been visited twice. Thus a loop exists, and the graph is not a DAG, a contradiction. Use the same argument to prove at least one source node exists. □

Lemma 3 *A connected CBT has an unique core node.*

Proof. Let the CBT consist of n nodes. The core node of an CBT must be a sink. By Lemma 2, an CBT has at least one sink. As the CBT is connected, there exist $n - 1$ uplinks. Map each uplink to the node it emanates from. Neither of these $n-1$ nodes is a sink as it has an emanating uplink. Then the only remaining node must be a sink and is therefore the unique core. □

Proposition 6 (Similarity between CBT and Unicycle) *Let* $T = (U, H)$ *be a weakly-connected digraph. Let for each* $u \in U$ *be* $\#upl(u) \leq 1$*. Then* U *is either a CBT or a unicycle.*

[4] *A directed graph. In a digraph the edges have a beginning and an ending node.*

[5] *For a directed graph, the underlying graph is its undirected version, i.e. the same graph with the edge directions dropped.*

From the Proposition 6 it is seen that the properties of T that the CBT algorithm must take care of are:

1. Ensure T is weakly-connected;
2. Ensure that for each $u \in U$, $\#upl(u) \leq 1$; and
3. Ensure that there exists at least one $u \in U$ that can locally decide whether T is a CBT or a unicycle.

The Meet Protocol

The meet protocol is first executed when node neighbourhood changes. Before any message exchange can occur between a pair of new neighbours, the conversation initiator must be established. This prevents the occurrence of a synchronization problem known as *crosstalk*, whereby both nodes either take the initiative or both nodes wait for the other to start. In Figure 6.5, the PN that realizes the *Meet* with crosstalk detection is shown[6]. The place and transition abbreviations are as follows: U: Undetermined (The state of the agent is undetermined: it is not yet known whether the node must assume the role of the Client or the Server); sr: Send Request; PA: Pending Answer; RR: Received Request; arc: Accept Role Client; rc: Role Conflict; CR: Conflicted roles mean that there is a (temporary) conflict between roles of A and B); vc: vote for client – this agent has decided to become a client; vs: vote for server – this agent has decided to become a server; PAC: Peer Accepted Client – this agent has been notified that its peer has accepted to become a client. Thus becoming the server is safe. apc: Acknowledge Peer as Client – acknowledge that the peer agent assumed the role of a Client in the interaction; RC: Role Client – the agent has accepted the role of a client; RS: Role Server – the agent has accepted the role of a server.

As the Figure 6.5 shows the interaction of two agents, the places and transitions in this diagram and onwards are annotated with a lower index A or B, to emphasize the agent to which they belong. The indices will be dropped in the later CPN figures if the distinction is not relevant or is handled in another way[7]. Further, throughout the diagrams, the same labels are used for one and the same place. Hence, the given CPN fragments can be attached together to form the complete CPN by overlapping all the places bearing identical names.

Theorem 6 (Crosstalk Detection) *Let Agents A and B execute the* Meet *protocol as given in the network $\Sigma_{6.5}$ in Figure 6.5. The following holds:*

$$\Sigma_{6.5} \vdash U_A\, U_B \hookrightarrow (RC_A \neg RS_A \neg RC_B\, RS_B) \vee (\neg RC_A\, RS_A\, RC_B \neg RS_B). \quad (6.1)$$

[6]Refer to Chapter 2 for the details of the CPN notation.
[7]This distinction can also be handled by means of *folding*.

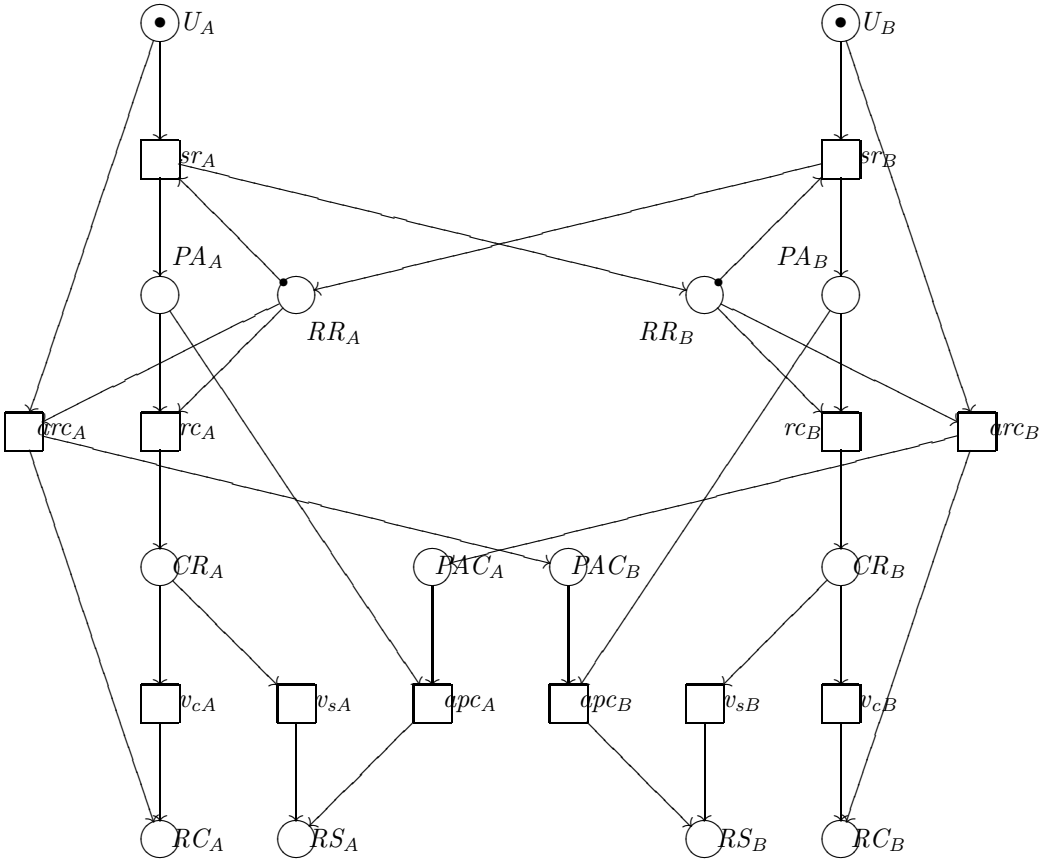


Figure 6.5: The Petri Net diagram of the *Meet* protocol between two agents, A and B.

Proof. The protocol unfolds until a decision about the role assignment has been reached. We first inspect the proof graph of finite length, given in Figure 6.6. The proof graph shows the *causes* relations that hold between the states of the CPN from the Figure 6.6. As the given CPN is symmetrical with respect to the permutation of agents A and B, we need to consider only one of the possible two first transitions. That is, in the Figure, only the firing of the transition sr_B is considered, as the case in which sr_A occurs gives the same proof graph, but with the agent names A and B interchanged.

Knowing this, the following reasoning leads to the proof of Equation (6.1). The derivation steps use the basic CPN triggering rules, which follow immediately from $\Sigma_{6.5}$. The symmetry of $\Sigma_{6.5}$ is used throughout.

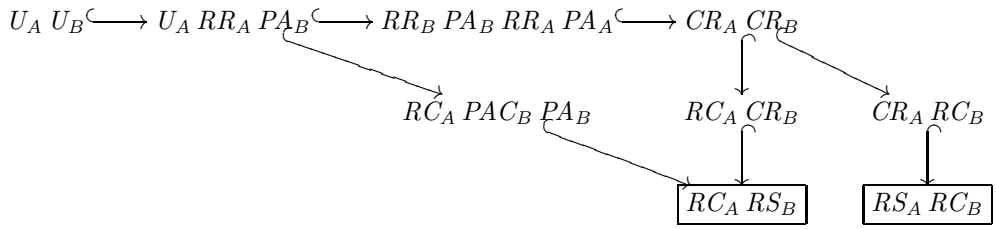


Figure 6.6: The crosstalk detection proof graph in the *Meet* protocol, for the network $\Sigma_{6.5}$.

First note that due to the CPN activation rules and the sequence of transitions leading from U_A to RC_A and RS_A, the following holds:

$$\Sigma_{6.5} \vdash U_A + PA_A + CR_A + RC_A + RS_A = 1 \Rightarrow RC_A + RS_A \leq 1. \tag{6.2}$$

The right-hand side of the rule states the mutual exclusion of the *client* and *server* states of A. By symmetry, the analogous claim holds for the B-part of $\Sigma_{6.5}$:

$$\Sigma_{6.5} \vdash RC_B + RS_B \leq 1. \tag{6.3}$$

Consider now the state in which $\Sigma_{6.5} \vdash RC_A$ holds. There are two ways this might have happened, either via arc_A, or via v_{cA}. Therefore the following holds:

$$\Sigma_{6.5} \vdash RC_A \Rightarrow (PAC_B + RS_B = 1) \vee (U_B + PA_B + CR_B + RS_B = 1).$$

The second part of the right hand side due to the assumed consensus between v_{cA} and v_{sB}, which is externally ensured through voting. Assuming progress on all transitions, eventually either $PAC_B = 0$, or $U_B + PA_B + CR_B = 0$, yielding in both cases:

$$\Sigma_{6.5} \vdash RC_A \Rightarrow RS_B = 1 \Leftrightarrow 0 \leq RC_A \leq RS_B \leq 1. \tag{6.4}$$

Substituting from here RC_A instead of RS_B into Equation (6.3), we obtain:

$$\Sigma_{6.5} \vdash RC_A + RC_B \leq 1, \tag{6.5}$$

proving mutual exclusion of the decisions in which both agents decide to be clients. Further, from adding Equation (6.2) and Equation (6.3), and taking into account Equation (6.5), one obtains

$$\Sigma_{6.5} \vdash RC_A + RC_B + RS_A + RS_B \leq 1 + RS_A + RS_B \leq 2, \tag{6.6}$$

from where we get:

$$\Sigma_{6.5} \vdash RS_A + RS_B \leq 1, \tag{6.7}$$

representing the mutex condition for the *server* decision for both agents. Finally we have:

$$\begin{aligned}\Sigma_{6.5} \vdash RS_A{+}RS_B \leq 1 \wedge RC_A + RC_B \leq 1 \\ \Rightarrow (RC_A \neg\, RS_A \neg\, RC_B\, RS_B) \vee (\neg\, RC_A\, RS_A\, RC_B \neg\, RS_B),\end{aligned} \tag{6.8}$$

thus proving the claim of Theorem 6. □

The Vote Protocol

The *Vote* protocol determines for a pair of nodes which one will be required to yield. The node that loses the vote is required to notify the core of its corresponding component that it must be disbanded i.e. that it must *yield* its functions to another (Figure 6.7). The vote requires both nodes in the pair to agree on a value

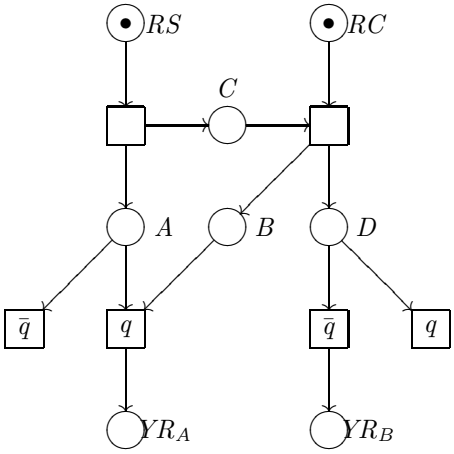


Figure 6.7: The *Vote* protocol.

of some predicate, shown as q in the $\Sigma_{6.7}$. As the outcome of a vote, only one of the nodes may start the *yield* request. Hence, only one of the two yield requests (shown in Figure 6.7 as YR_A and YR_B) may be activated.

Theorem 7 (Vote) *Let there be given a PN as in $\Sigma_{6.7}$. Then the following holds:*

$$\Sigma_{6.7} \vdash YR_A + YR_B \leq 1.$$

Proof. From $\Sigma_{6.7}$, the following hold:

$$\begin{aligned}\Sigma_{6.7} &\vdash q \Rightarrow YR_B = 0 \wedge RS + A + YR_A = 1 \\ \Sigma_{6.7} &\vdash \bar{q} \Rightarrow YR_A = 0 \wedge RC + D + YR_B = 1,\end{aligned}$$

since $q = \text{true}$ implies that the transition leading to YR_B never executes, and that the transition sequence leading to YR_A must occur eventually (and vice-versa, for $q = \text{false}$). Hence, first:

$$\frac{\dfrac{\Sigma_{6.7} \vdash q \Rightarrow\ YR_B = 0 \wedge RS + A + YR_A = 1}{\Sigma_{6.7} \vdash q \Rightarrow\ YR_B = 0 \wedge\ YR_A \leq 1}\ [\ RS \leq 1 \wedge A \leq 1\ ]}{\Sigma_{6.7} \vdash q \Rightarrow\ YR_A + YR_B \leq 1.}$$

Thereafter, similarly:

$$\frac{\dfrac{\Sigma_{6.7} \vdash \bar{q} \Rightarrow\ YR_A = 0 \wedge RS + C + D + YR_B = 1}{[RC \leq 1 \wedge C \leq 1 \wedge D \leq 1]\Sigma_{6.7} \vdash \bar{q} \Rightarrow\ YR_B = 0 \wedge\ YR_A \leq 1}}{\Sigma_{6.7} \vdash \bar{q} \Rightarrow\ YR_A + YR_B \leq 1.}$$

Finally, collecting the two conclusions:

$$\frac{\begin{array}{l}\Sigma_{6.7} \vdash q \Rightarrow\ YR_A + YR_B \leq 1\\ \Sigma_{6.7} \vdash \bar{q} \Rightarrow\ YR_A + YR_B \leq 1\end{array}}{[(\varphi \Rightarrow \mu \wedge \bar{\varphi} \Rightarrow \mu) \Rightarrow \mu]\Sigma_{6.7} \vdash\ YR_A + YR_B \leq 1.}$$

Also note that $\Sigma_{6.7} \vdash\ YR_A + YR_B = 1 \hookrightarrow RC = 0 \wedge RS = 0$ assumes progress, i.e. it is assumed the said transition eventually occurs. □

The Switch Protocol

In this section, claims are made about the environment in which the nodes (and the co-located agents) operate, as well as the internal soft states of the nodes. To achieve this, all the nodes must be considered at once. This is achieved by *folding* (see [69]) all the local CPNs into one diagram, rather than considering the local CPNs, as follows. If for some node n its internal place Q has a token t (denoted as $Q.t$ at n), that token can be *folded* to a pair (n, t) so that $Q.(n, t)$ can be written to denote the same fact. The discussion of folding is left at that for the purpose of this Chapter as more details can be obtained from [69].

In the switch protocol, two nodes that did not have connection to each other, nor it is immediately possible to figure out whether they belong to the same component, need to agree which node adopts a new uplink to the other one. This is called a *yield*. Only a core can yield, as only to the core can a new uplink be added. If a yield is requested from a non-core node, then the function of the core must first be moved to it, before the yield becomes possible. The motion is realized in a sequence of *switches* (see Figure 6.3 for a reminder). The switch protocol is divided for simplicity into two parts, called the *top* and the *bottom* half (see Figure 6.8 and Figure 6.9 respectively). The top half is used to propagate

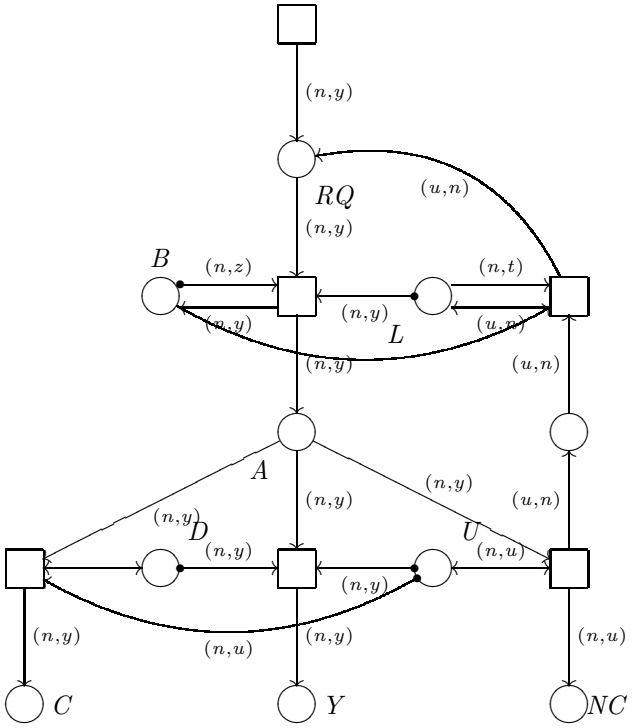


Figure 6.8: Top half of the Switch protocol.

a yield request to the core, and to initiate switches in the appropriate sequence. The bottom half is concerned with the concurrent execution of a single switch.

Before considering the protocol details, it is in order to explain the motivation of the protocol. The switch protocol is used to connect two CBTs into a single CBT. To preserve the CBT invariant of Proposition 5, one of the two core nodes must yield its function to the other. This can only happen upon a join, and as the yield function can only be performed by a core node, it follows that if a non-core node is selected to yield, it first has to become core and only then can it yield. The non-core node can issue a request to its uplink to move the core functionality closer (in number of hops through the CBT). If its uplink is a non-core too, it must propagate the query to its uplink until the core is eventually reached. When the core is reached, it decides locally which of the requests is it going to honor, and switches to its downlink that propagated the honored request. Thereby the roles of the downlink and the uplink are exchanged and the core effectively moves.

It must be proven that in the switch protocol, any request for a yield is eventually followed by a yield. Three cases need to be considered:

1. A core is requested a switch from a non-downlink (i.e. a non-connected

node);

2. A core is requested a switch from its downlink; or

3. A switch was requested from a non-core node.

The first is the base case, whereby when a switch is requested to a node which is already the core, it can proceed to yield immediately. The second case is when a single switch occurs, whereby the core is moved from its old place to that of the honoured request's downlink. The third case is when a non-core node is requested to switch, either by a downlink, or by a new node. In the third case, the task of the non-core node is to propagate the request further to the uplink. Before formulating the final theorem, a three-part lemma about the switching is formulated as follows.

Lemma 4 (Top Half Switch Protocol) *Let the labels a, b and u denote nodes in the set V of $\Gamma(t) = (V, E)$. Where appropriate, assume that $u = upl(a)$.*

1. *Let $W_1 == \forall\, q \bullet \neg U.(a, q) \wedge \neg D.(a, b)$, $W_2 == \neg L.(a, b) \wedge \neg B.(a, b)$ and $W == W_1 \wedge W_2$ be the context. Consider the CPN of $\Sigma_{6.8}$. Then:*

$$W \vdash RQ.(a, b) \hookrightarrow Y.(a, b).$$

2. *Let $W_3 == D.(a, b) \wedge U.(b, a) \wedge \forall\, q \bullet \neg U.(a, q)$ and $W_4 == D.(a, b) \wedge U.(b, a) \wedge \forall\, q \bullet \neg U.(a, q)$. Let $W == W_3 \wedge W_4$ be the context. Then:*

$$W \vdash RQ.(a, b) \wedge D.(a, b) \wedge U.(b, a) \hookrightarrow C.(a, b).$$

3. *Let the context be: $W == U.(a, u) \wedge \neg D.(a, b) \wedge \neg D.(b, a) \wedge \neg B.(a, b)$. Then:*

$$W \vdash RQ.(a, b) \hookrightarrow NC.(a, u) \wedge B.(a, b) \wedge C.(u, a).$$

Proof. The lemma components are proven in turn.

1. Let a be the core node and b be a non-downlink. Hence the following assumptions hold.

$$W_1 == \forall\, q \bullet \neg U.(a, q) \wedge \neg D.(a, b).$$

For simplicity, it is further considered that

$$W_2 == \neg L.(a, b) \wedge \neg B.(a, b)$$

and

$$W == W_1 \wedge W_2$$

as the assumption. Considering $\Sigma_{6.8}$, it is proven:

$$\frac{\frac{\frac{W \vdash RQ.(a,b)}{RQ.(a,b) \wedge W \hookrightarrow A.(a,b) \wedge W}}{[\neg D.(a,b) \wedge A.(a,b) \wedge \neg U.(a,b)]RQ.(a,b) \wedge W \hookrightarrow Y.(a,b) \wedge W}}{W \vdash RQ.(a,b) \hookrightarrow Y.(a,b).}$$

2. Consider the case where b is the downlink of a, and a is the core. Then assume:

$$W_3 == D.(a,b) \wedge U.(b,a) \wedge \forall\, q \bullet \neg U.(a,q).$$

Again for simplicity:

$$W_4 == \forall\, q \bullet \neg B.(a,q) \wedge \neg L.(a,q),$$

so that now the context is:

$$W == W_3 \wedge W_4.$$

This yields:

$$\frac{\frac{W \vdash RQ.(a,b) \hookrightarrow A.(a,b)}{W \vdash RQ.(a,b) \hookrightarrow C.(a,b)}}{W \vdash RQ.(a,b) \wedge D.(a,b) \wedge U.(b,a) \hookrightarrow C.(a,b),}$$

as required.

3. Consider the case where a and b are not connected, i.e. there is no $(a,b) \in E$, and a has an uplink u. The context is now:

$$W == U.(a,u) \wedge \neg D.(a,b) \wedge \neg D.(b,a) \wedge \neg B.(a,b).$$

Then:

$$\frac{\frac{\frac{W \vdash RQ.(a,b) \hookrightarrow A.(a,b) \wedge B.(a,b)}{[U.(a,u)]\, W \vdash RQ.(a,b) \hookrightarrow NC.(a,u) \wedge W.(u,a) \wedge B.(a,b)}}{W \vdash RQ.(a,b) \hookrightarrow NC.(a,u) \wedge L.(u,a) \wedge RQ.(u,a) \wedge B.(a,b)}}{W \vdash RQ.(a,b) \hookrightarrow NC.(a,u) \wedge RQ.(u,a) \wedge B.(a,b).}$$

Thus, it is seen that a request from a non-downlink b to the node a leaves the node a busy with processing the request from b (token $B.(a,b)$), and places it in the waiting queue for the switch for u (token $NC.(a,u)$). Applying Item 2 it is obtained:

$$\frac{W \vdash RQ.(a,b) \hookrightarrow NC.(a,u) \wedge RQ.(u,a) \wedge B.(a,b)}{W \vdash RQ.(a,b) \hookrightarrow NC.(a,u) \wedge B.(a,b) \wedge C.(u,a),} \quad [\ \textit{Item}\ 2\ ]$$

which is the condition for the entry to the bottom half of the switch protocol. □

Lemma 5 (Bottom Half Switch Protocol) *Let the labels a, b and u denote nodes in the set V of $\Gamma(t) = (V,E)$. Where appropriate, it is assumed that $u = upl(a)$. Then:*

$$NC.(a,u) \wedge C.(u,a) \wedge B.(a,b) \wedge D.(u,a) \wedge U.(a,u) \hookrightarrow \\ D.(a,u) \wedge U.(u,a).$$

Proof.

$$\frac{NC.(a,u) \wedge C.(u,a) \wedge B.(a,b) \wedge D.(u,a) \wedge U.(a,u) \hookrightarrow \\ M.(\mathsf{T},u,a) \wedge N.\langle \mathsf{rq},u,a,(\mathsf{p},\mathsf{T})\rangle \\ \wedge NC.(a,u) \wedge C.(u,a) \wedge B.(a,b) \wedge B.(u,a) \wedge D.(u,a) \wedge U.(a,u)}{NC.(a,u) \wedge C.(u,a) \wedge B.(a,b) \wedge D.(u,a) \wedge U.(a,u) \hookrightarrow \\ M.(\mathsf{T},u,a) \wedge [(x \wedge O.\langle \mathsf{rf},a,u,\mathsf{T}\rangle) \vee (\bar{x} \wedge P.\langle \mathsf{ac},a,u,\mathsf{T}\rangle \wedge Q.(\mathsf{T},a,u))] \\ \wedge B.(a,b) \wedge B.(u,a) \wedge D.(u,a) \wedge U.(a,u).}$$

The latter inference can be split depending on the value of the predicate x. The only interesting case is when x is false (denoted by $\bar{x}$), when nodes proceed with the switch. Within the context where $\bar{x}$ holds, it can be derived:

$$\frac{\bar{x} \vdash NC.(a,u) \wedge C.(u,a) \wedge B.(a,b) \wedge D.(u,a) \wedge U.(a,u) \hookrightarrow \\ S.(\mathsf{T},u,a) \wedge R.\langle \mathsf{rq},u,n,\mathsf{c},\mathsf{T}\rangle \\ \wedge Q.(\mathsf{T},a,u) \wedge B.(a,b) \wedge B.(u,a) \wedge D.(u,a) \wedge U.(a,u)}{\bar{x} \vdash NC.(a,u) \wedge C.(u,a) \wedge B.(a,b) \wedge D.(u,a) \wedge U.(a,u) \hookrightarrow \\ D.(a,u) \wedge U.(u,a).}$$

Considering that the predicate x is external (i.e. unrelated to the decision process), consider only the context $\bar{x}$ is considered. Considering only that context (i.e. assuming that the switch is not externally invalidated) yields the claim. □

Equipped with Lemmata 4 and 5 the main switch theorem can be stated.

Theorem 8 (Switch Theorem) *Let a and b be nodes of V, in $\Gamma(t) = (V,E)$ such that $(a,b) \notin E$. Let $u = upl(a)$. Then the following holds:*

$$RQ.(a,b) \hookrightarrow Y.(a,b) \wedge B.(a,b).$$

Theorem 8 notes that in any CBT, and any two nodes a and b that do not obviously belong to the same CBT, a request for a join is eventually followed by activating the *yield* protocol.

Proof. By induction. The base case is Item 1 of Lemma 4. Adopting an inductive hypothesis from Lemma 4 (items 2 and 3) and Lemma 5, it is seen that this happens when u is also the core.

Now consider (with W as in item 3 of Lemma 4):

$$\frac{\begin{array}{l} W \vdash RQ.(a,b) \hookrightarrow RQ.(a,b) \wedge NC.(a.u) \wedge RQ.(u,a) \wedge B.(a,b) \\ \hline W \vdash RQ.(a,b) \hookrightarrow RQ.(a,b) \wedge NC.(a,u) \wedge C.(u,a) \wedge B.(a,b) \\ \hline RQ.(a,b) \wedge W \hookrightarrow RQ.(a,b) \wedge D.(a,u) \wedge U.(u,a) \wedge B.(a,b) \\ \hline RQ.(a,b) \wedge W \hookrightarrow Y.(a,b) \wedge D.(a,u) \wedge U.(u,a) \wedge B.(a,b) \\ \hline W \vdash RQ.(a,b) \hookrightarrow Y.(a,b) \wedge B.(a,b) \end{array}}{\begin{array}{l} [\ L.\ 5\ ] \\ [\ L.,\ 4it.\ 1\ ] \end{array}}$$

which was to be shown. The remainder of the join is handled by the *Yield* protocol. □

The Yield Protocol

The role of the Yield protocol is to attach two CBTs together.

Theorem 9 (Yield Protocol) *Let $\Gamma(t) = (V, E)$ be a connectivity graph at t. Consider the protocol given in $\Sigma_{6.10}$, and two nodes $a, b \in V$. Then the following holds.*

$$\begin{array}{l} R.(b,a) \wedge B.(b,a) \wedge CID.(a, \mathsf{C}_a) \wedge CID.(b, \mathsf{C}_b) \wedge Y.(b,a) \\ \quad \hookrightarrow (\mathsf{C}_a = \mathsf{C}_b) \vee [(\mathsf{C}_a \neq \mathsf{C}_b) \wedge U.(b,a) \wedge D.(a,b) \wedge CID.(b, \mathsf{C}_a)]. \end{array}$$

Proof. Throughout the proof assume that in $\Sigma_{6.10}$ the theorem:

$$Q.y \hookrightarrow A.(y, \mathsf{C}) \wedge CID.(coreof(y), \mathsf{C})$$

holds, i.e. that the node y in question can obtain the component identifier from the core of its component. This is justified later by Theorem 10. Adopt a shorthand:

$$W == R.(b,a) \wedge B.(b,a) \wedge CID.(a, \mathsf{C}_a) \wedge CID.(b, \mathsf{C}_b) \wedge Y.(b,a).$$

Then, from $\Sigma_{6.10}$, the following holds:

$$W \hookrightarrow R.(b,a) \wedge B.(b,a) \wedge CID.(a,\mathsf{C}_a) \wedge F.(b,a,\mathsf{T}) \wedge E.\langle \mathsf{rq}, b, a, (\mathsf{p},\mathsf{T},\mathsf{C}_b)\rangle$$

$$W \hookrightarrow R.(b,a) \wedge B.(b,a) \wedge CID.(a,\mathsf{C}_a) \wedge F.(b,a,\mathsf{T}) \wedge P.(b,a,(\mathsf{p},\mathsf{T},\mathsf{C}_b)) \wedge Q.a$$

$$W \hookrightarrow R.(b,a) \wedge B.(b,a) \wedge CID.(a,\mathsf{C}_a) \wedge F.(b,a,\mathsf{T}) \wedge P.(b,a,(\mathsf{p},\mathsf{T},\mathsf{C}_b)) \wedge A.(a,\mathsf{C}_a)$$

$$W \hookrightarrow R.(b,a) \wedge B.(b,a) \wedge CID.(a,\mathsf{C}_a) \wedge F.(b,a,\mathsf{T}) \wedge G.(a,b,\mathsf{T},\mathsf{p},\mathsf{C}_b,\mathsf{C}_a)$$

The predicate φ locally compares two component identifiers, in this case C_a and C_b. It is *true* whenever the component identifiers are equal. Hence:

$$W \hookrightarrow R.(b,a) \wedge B.(b,a) \wedge CID.(a,\mathsf{C}_a) \wedge F.(b,a,\mathsf{T}) \wedge [(H.\langle \mathsf{rf}, b, a, (\mathsf{p},\mathsf{T})\rangle \wedge \mathsf{C}_a = \mathsf{C}_b) \vee (I.\langle \mathsf{ac}, b, a, (\mathsf{p},\mathsf{T})\rangle \wedge \mathsf{C}_a \neq \mathsf{C}_b \wedge K.(a,b,\mathsf{T}))].$$

Two cases are considered now, depending on the value of φ. First consider the case when $\mathsf{C}_a = \mathsf{C}_b$, and afterwards when $\mathsf{C}_a \neq \mathsf{C}_b$:

$$\mathsf{C}_a = \mathsf{C}_b \vdash W \hookrightarrow R.(b,a) \wedge B.(b,a) \wedge CID.(a,\mathsf{C}_a) \wedge F.(b,a,\mathsf{T}) \wedge H.\langle \mathsf{rf}, b, a, (\mathsf{p},\mathsf{T})\rangle$$

$$\mathsf{C}_a = \mathsf{C}_b \vdash W \hookrightarrow CID.(a,\mathsf{C}_a)$$

$$W \hookrightarrow CID.(a,\mathsf{C}_a) \wedge \mathsf{C}_a = \mathsf{C}_b.$$

Also:

$$\mathsf{C}_a \neq \mathsf{C}_b \vdash W \hookrightarrow R.(b,a) \wedge B.(b,a) \wedge CID.(a,\mathsf{C}_a) \wedge F.(b,a,\mathsf{T}) \wedge I.\langle \mathsf{ac}, b, a, (\mathsf{p},\mathsf{T})\rangle \wedge K.(a,b,\mathsf{T})$$

$$\mathsf{C}_a \neq \mathsf{C}_b \vdash W \hookrightarrow R.(b,a) \wedge B.(b,a) \wedge CID.(a,\mathsf{C}_a) \wedge J.(b,a,\mathsf{T}) \wedge K.(a,b,\mathsf{T}) \wedge L.\langle \mathsf{rq}, b, a, (\mathsf{c},\mathsf{T})\rangle$$

$$\mathsf{C}_a \neq \mathsf{C}_b \vdash W \hookrightarrow R.(b,a) \wedge B.(b,a) \wedge CID.(a,\mathsf{C}_a) \wedge J.(b,a,\mathsf{T}) \wedge M.\langle \mathsf{ac}, a, b, (\mathsf{c},\mathsf{T})\rangle \wedge D.(a,b) \wedge CID.(b,\mathsf{C}_a)$$

$$\mathsf{C}_a \neq \mathsf{C}_b \vdash W \hookrightarrow CID.(a,\mathsf{C}_a) \wedge CID.(b,\mathsf{C}_a) \wedge D.(a,b) \wedge U.(b,a)$$

$$\mathsf{C}_a \neq \mathsf{C}_b \vdash W \hookrightarrow CID.(b,\mathsf{C}_a) \wedge D.(a,b) \wedge U.(b,a)$$

$$W \hookrightarrow CID.(b,\mathsf{C}_a) \wedge D.(a,b) \wedge U.(b,a) \wedge \mathsf{C}_a \neq \mathsf{C}_b.$$

Joining the above two inferences, the claim of the theorem is obtained.

$$\frac{\begin{array}{l} W \hookrightarrow CID.(b, \mathsf{C}_a) \wedge D.(a, b) \\ \quad \wedge\ U.(b, a) \wedge \mathsf{C}_a \neq \mathsf{C}_b \\ W \hookrightarrow CID.(a, \mathsf{C}_a) \wedge \mathsf{C}_a = \mathsf{C}_b \end{array}}{\begin{array}{l} W \hookrightarrow (\mathsf{C}_a = \mathsf{C}_b) \vee [(\mathsf{C}_a \neq \mathsf{C}_b) \wedge U.(b, a) \\ \quad \wedge\ D.(a, b) \wedge CID.(b, \mathsf{C}_a)] \end{array}}$$

□

Obtaining Component Identifier

Obtaining the component identifiers is used to determine whether two joining nodes should commit the join transaction. It serves two purposes. First, checks for the attempts of joins within the same component. Second, it implicitly removes the cycle in the case of unicycle underlying graph. The Figure 6.11 shows the component identifier algorithm. In Figure 6.10, this protocol component has been compactly represented by the sequence (Q, cq, A).

Theorem 10 (Component Identifier) *Let $\Gamma(t) = (V, E)$ be a connectivity graph at t. Let $T = (V, U)$ be a CBT. Let n and u be arbitrary elements from V, such that $upl(n) = u$. Then the following holds:*

$$T \vdash Q.n \hookrightarrow A.(n, \mathsf{C}) \wedge (CID.(coreof(n), \mathsf{C}) \vee (n \notin dom\, upl \wedge CID.(n, \mathsf{C}))).$$

Theorem 10 says that a component identifier query on the place Q at node n causes the answer on A with the component identifier C. Then C is either that of the core node corresponding to n, or is own identifier if n is the core.

Proof. By induction. It is required to establish that the following hold. After proving each step, all are assembled into the final proof.

1. *Base case.* For the base case assume n is the unique core node, i.e. $n \notin \mathrm{dom}\, upl$. Then:

$$T, n \notin \mathrm{dom}\, upl \vdash Q.n \hookrightarrow A.(coreof(n), \mathsf{C}) \wedge CID.(n, \mathsf{C}).$$

2. *Inductive Hypothesis.* The query in the uplink causes an answer in the uplink.

$$T \vdash PQ.(u, P, r) \hookrightarrow AQ.(u, P, \mathsf{C}) \wedge CID.(coreof(u), \mathsf{C}).$$

3. *Inductive Step.* The inductive step consists of three sub-parts: the *upward propagation*, the *downward propagation* and finally the *answer*.

$$T, n \in \mathrm{dom}\, upl \vdash Q.n \hookrightarrow PQ.(u, P \cup \{n\}, r) \wedge PQ.(n, Q \cup \{n\}, q)$$
$$T, n \in \mathrm{dom}\, upl \vdash AQ.(u, P \cup \{n\}, \mathsf{C}) \wedge PQ.(n, Q, q) \hookrightarrow AQ.(n, Q, \mathsf{C})$$
$$T, n \in \mathrm{dom}\, upl \vdash AQ.(n, P \cup \{n\}, \mathsf{C}) \hookrightarrow A.(n, \mathsf{C}).$$

Each of the above items is proven in turn. For the base case assume $CID.(u, \mathsf{C})$. From $\Sigma_{6.11}$, it is immediate that:

$$\frac{Q.n \hookrightarrow R.(n,n) \hookrightarrow \overbrace{PQ(n,\{n\},r) \hookrightarrow AQ.(n,\{n\},\mathsf{C})}^{a} \hookrightarrow A.(n,\mathsf{C})}{[CID.(n,\mathsf{C})]\, Q.n \hookrightarrow A.(n,\mathsf{C}) \wedge CID.(n,\mathsf{C}).}$$

Also:

$$T, n \notin \operatorname{dom} upl \vdash Q.n \hookrightarrow A.(n,\mathsf{C}) \hookrightarrow AQ.(n,\{n\},\mathsf{C}) \hookrightarrow A.(n,\mathsf{C}).$$

For the inductive hypothesis, note first that the element a from the above derivation establishes a case in which the inductive hypothesis holds. Thus, the hypothesis is valid.

For the upward propagation the following derivation holds:

$$\frac{Q.n \hookrightarrow PQ.(n, P \cup \{n\}, r)}{\begin{array}{l}[u = upl(n) \Rightarrow U.(n,u)]\, Q.n \hookrightarrow PQ.(n, P \cup \{n\}, r) \wedge U.(n,u) \\ PQ.(n,P,r) \wedge PQ.(u,Q,q) \\ \quad \wedge\, U.(n,u) \Rightarrow PQ.(u, Q \cup \{n\}, q)\end{array}}$$

$$\frac{}{[PQ.(u,Q,q)]\, Q.n \hookrightarrow PQ.(n, P \cup \{n\}, r)}$$

$$\frac{}{\begin{array}{l}[u = upl(n) \Rightarrow U.(n,u)]\, Q.n \hookrightarrow PQ.(n, P \cup \{n\}, r) \\ \quad \wedge\, PQ.(u, Q \cup \{n\}, min(q,r)) \\ \quad \wedge\, U.(n,u)\end{array}}$$

$$\frac{}{[p \wedge q \Rightarrow q,\ t := min(q,r)]\, Q.n \hookrightarrow PQ.(n, P \cup \{n\}, r) \wedge PQ.(u, Q \cup \{n\}, t).}$$

For the answer:

$$T, n \in \operatorname{dom} upl \vdash AQ.(n, P \cup \{n\}, \mathsf{C}) \hookrightarrow A.(n,\mathsf{C})$$

proceed by induction on $\#P$:

1. *Base case.* If $\#P = 0$, from $\Sigma_{6.11}$ the following is available immediately:

$$\frac{AQ.(n,\{n\},\mathsf{C})}{[AQ.(n,\{n\},\mathsf{C}'') \hookrightarrow A.(n,\mathsf{C}'')]\, AQ.(n,\{n\},\mathsf{C}) \hookrightarrow A.(n,\mathsf{C})}$$

2. *Inductive Hypothesis.* Assume that for all $\#P = k - 1$ the theorem holds.

3. *Inductive Step.*

$$\frac{AQ.(n, P \cup \{n\}, \mathsf{C})}{AQ.(n, P \cup \{n\}, \mathsf{C}) \hookrightarrow \exists\, y \bullet AQ.(n, R \cup \{y, n\}, \mathsf{C}) \wedge PQ.(y, S, t)} \; [\text{ upw. prop. }]$$

$$\frac{AQ.(n, P \cup \{n\}, \mathsf{C}) \hookrightarrow \exists\, y \bullet AQ.(n, R \cup \{y, n\}, \mathsf{C}) \wedge PQ.(y, S, t)}{AQ.(n, P \cup \{n\}, \mathsf{C}) \hookrightarrow AQ.(n, R \cup \{n\}, \mathsf{C}) \wedge AQ.(y, Q, \mathsf{C})} \; [\text{ by trans. } d \text{ }]$$

$$\frac{AQ.(n, P \cup \{n\}, \mathsf{C}) \hookrightarrow AQ.(n, R \cup \{n\}, \mathsf{C}) \wedge AQ.(y, Q, \mathsf{C})}{AQ.(n, P \cup \{n\}, \mathsf{C}) \hookrightarrow AQ.(n, R \cup \{n\}, \mathsf{C})} \; [\text{ } p \wedge q \Rightarrow p \text{ }]$$

$$\frac{AQ.(n, P \cup \{n\}, \mathsf{C}) \hookrightarrow AQ.(n, R \cup \{n\}, \mathsf{C})}{AQ.(n, P \cup \{n\}, \mathsf{C}) \hookrightarrow A.(n, \mathsf{C}).} \; [\text{ ind. hyp. }]$$

The proof components can now be assembled.

$$\frac{Q.n \hookrightarrow PQ(u, P \cup \{n\}, r) \wedge PQ.(n, Q \cup \{n\}, q)}{Q.n \hookrightarrow AQ.(u, P \cup \{n\}, \mathsf{C}) \wedge \mathit{CID}.(\mathit{coreof}\,(u), \mathsf{C}) \wedge PQ.(n, Q \cup \{n\}, q)}$$

$$\frac{Q.n \hookrightarrow AQ.(u, P \cup \{n\}, \mathsf{C}) \wedge \mathit{CID}.(\mathit{coreof}\,(u), \mathsf{C}) \wedge PQ.(n, Q \cup \{n\}, q)}{Q.n \hookrightarrow AQ.(n, Q \cup \{n\}, \mathsf{C}) \wedge \mathit{CID}.(\mathit{coreof}\,(u), \mathsf{C})}$$

$$\frac{Q.n \hookrightarrow AQ.(n, Q \cup \{n\}, \mathsf{C}) \wedge \mathit{CID}.(\mathit{coreof}\,(u), \mathsf{C})}{Q.n \hookrightarrow A.(n, \mathsf{C}) \wedge \mathit{CID}.(\mathit{coreof}\,(u), \mathsf{C})}$$

$$\frac{Q.n \hookrightarrow A.(n, \mathsf{C}) \wedge \mathit{CID}.(\mathit{coreof}\,(u), \mathsf{C})}{Q.n \hookrightarrow A.(n, \mathsf{C}) \wedge \mathit{CID}.(\mathit{coreof}\,(n), \mathsf{C}).}$$

From the base case and the inductive proof, as required:

$$\frac{T \vdash Q.n \hookrightarrow A.(n, \mathsf{C}) \wedge \mathit{CID}.(\mathit{coreof}\,(n), \mathsf{C}) \wedge n \in \operatorname{dom} \mathit{upl} \qquad T \vdash Q.n \hookrightarrow A.(n, \mathsf{C}) \wedge \mathit{CID}.(\mathit{coreof}\,(n), \mathsf{C}) \wedge n \notin \operatorname{dom} \mathit{upl}}{T \vdash Q.n \hookrightarrow A.(n, \mathsf{C}) \wedge (\mathit{CID}.(n, \mathsf{C}) \vee (n \notin \operatorname{dom} \mathit{upl} \wedge \mathit{CID}.(n, \mathsf{C}))).}$$

□

6.5 Performance

Two figures of merit are of interest for the performance of the CBT construction and maintenance algorithm. They are the *stretch* (the maximum path length between the nodes) of the constructed graph and the message complexity of the used protocol. The message complexity is expressed in terms of the number of the expected number of core switches needed to make a CBT with n nodes, as the number of messages per each core switch is constant. The stretch of a CBT is upper bounded by $O(n^c)$, and the message complexity by $O(n)$, where $c = \log_2 3 - 1$. In the analysis some simplifying assumptions are made. First, it is assumed that at some point in time, the maximal CBT is constructed. In deployment, the CBT construction never finishes as long as there are changes in the network. Instead, it continuously repairs the CBT so that only a maximal one remains, provided that this is allowed by the network dynamics. Second, the concurrent requests to join two components, coming from different pairs of nodes, as well as the cycle

breaking are all neglected. This is because the number of concurrent requests does not change the order of the number of switches required, although it does alter the constant. Furthermore, the number of concurrent requests depends on the number of nodes in different components that come into contact, a parameter outside of the scope of this analysis.

The following lemma is needed to prove complexity claims so it is stated first, before the claims and proofs themselves.

Lemma 6 *Let f and g be functions of a positive integer k, defined as:*

1. *$f(k) = 1/2 + 3/2\ f(\lceil k/2 \rceil)$, with $f(1) = 0$;*
2. *$g(k) = \lceil k/2 \rceil^c + 2g(\lceil k/2 \rceil)$, with $g(1) = 0$.*

Then, $f(k) = O(k^c)$ and $g(k) = O(k)$, where $c = \log_2 3 - 1$.

Proof. For both items, consider only $k = 2^p$ for some integer p (otherwise replace k by $2^{\lceil \log_2 k \rceil}$ and proceed).

For Item 1, expand f recursively to obtain:

$$f(k) = 1/2 \sum_{l=0}^{p-1} (3/2)^l = (3/2)^{\log_2 n} - 1 = O(n^c).$$

For Item 2, expand g recursively to obtain:

$$g(k) = \sum_{l=0}^{p-1} 2^l \cdot 2^{(p-l-1)c} = 2^{c(p-1)} \cdot [2^{(1-c)p} - 1]/(2^{1-c} - 1) = O(2^p) = O(k).$$

□

The expected stretch of a graph constructed by the CBT algorithm is now computed. Let T be a CBT constructed by our CBT algorithm, and let $\#T$ be the number of vertices in T. The CBT T was constructed by concurrent applications of the CBT construction and maintenance algorithm, and T itself was obtained by joining together two non-maximal CBTs, say U and W, by adopting an uplink between two nodes $x' \in U$ and $y' \in W$. For x and y two nodes in T, denote as $h_T(x, y)$ the hop count of the only path between x and y in T.

Theorem 11 *The expected stretch of the CBT T, with $\#T$ nodes is $O(\#T^c)$.*

Proof. The mean hop count l_T in T is $l_T = \#T^{-2} \cdot \sum_{x,y \in T} h_T(x, y)$. Also let $L_{\#T} = \mathrm{E}[l_T]$ for a CBT T with $\#T$ nodes.

Separating over hopcounts in U and W:

$$\begin{aligned} l_T &= \#T^{-2} \left\{ \sum_{x,y \in U} h_U(x, y) + \sum_{x,y \in W} h_W(x, y) \right. \\ &+ \left. 2 \left[\sum_{x \in U, y \in W} h_U(x, x') + 1 + h_W(y, y') \right] \right\}. \end{aligned} \tag{6.9}$$

Taking the expectation of the hop count (over all the possible graphs T, and also all components U and W), and by linearity of expectation:

$$\begin{aligned} L_{\#T} &= \#T^{-2}(\#U^2 L_{\#U} + \#W^2 L_{\#W} + 2\#U\#W \\ &+ 2\#U\#W L_{\#W} + 2\#U\#W L_{\#U}). \end{aligned} \tag{6.10}$$

As $\#T = \#U + \#W$, and $\#U, \#W > 0$, $\#U\#W/\#T^2 \leq 1/4$, hence rewriting Eq. (6.10):

$$\begin{aligned} L_{\#T} &\leq 1/2 + \frac{\#UL_{\#U} + \#WL_{\#W}}{\#T} + \frac{1}{4}(L_{\#U} + L_{\#W}) \\ &\leq 1/2 + 3/2\, max(L_{\#U}, L_{\#W}), \end{aligned} \tag{6.11}$$

where Eq. (6.11) holds over any partition of T to U and W. Hence it must also hold for $max(L_{\#U}, L_{\#W}) = L_{\lceil \#T/2 \rceil}$. Applying Item 1 of Lemma 6 to the right hand side of Eq. (6.11) for $\#U = \lceil \#T/2 \rceil$ finishes the proof. □

Theorem 12 *The expected number of core switches needed to construct a CBT T with $\#T$ nodes is $O(\#T)$.*

Proof. Let $M_{\#T}$ be the expected number of switches to construct a CBT T by connecting the smaller CBTs U and W. Total expected number of switches equals the expected number of switches used to connect U and W plus the expected number of switches to construct both U and W. Thus the recursive expression holds:

$$M_{\#T} = 1/2(L_{\#U} + L_{\#W}) + M_{\#U} + M_{\#W}, \tag{6.12}$$

where the first component is due to the uniform random choice of the component whose core executes the yield. As $M_{\#T}$ is nondecreasing, assume $\#U = \lceil \#T/2 \rceil$ to get:

$$M_{\#T} \leq L_{\lceil \#T/2 \rceil} + 2M_{\lceil \#T/2 \rceil} = O(\#T). \tag{6.13}$$

The last equality, due to Item 2 of Lemma 6, completes the proof. □

6.6 Summary

In this Chapter, we presented a distributed data structure used for service description forwarding, useful in distributed systems operating in the MANET environment. One must bear in mind that the CBT has been designed for MANETs, where each node has a relatively small number of neighbours and where it is typically difficult or not profitable to keep extensive records of the nodes far away in the network. This use case is hence different from the peer-to-peer networks which use the Internet. In peer-to-peer networks all pairs of neighbours can communicate and the problem is to find and use only the efficient number of connections. In MANET however, each node has only a few connections and the problem is to

inform the further nodes well by forwarding the right amount of soft state. The CBT is designed to achieve this.

We described the CBT structure, then gave detailed formal account how it is constructed and maintained at runtime and discussed its complexity. We presented the algorithm that uses the CBT structure to propagate the service descriptions. We addressed the concern of service description distribution strategy, with a distributed structure which is flat, requiring no dedicated nodes to operate, and requiring no preconfiguration. This is in contrast to service distribution schemes which use a backbone (as discussed in Section 6.1), where role divisions among nodes are implicit for the backbone nodes. Instead, each node in the CBT takes an equivalent role in the distribution, and due to the core mobility, is equally likely to become the core node. We do not consider the existence of the core node the departure from the flat distribution structure. This is because the core performs no special functions, except that it is the node to which all the uplinks point to.

The important property of the core's existence is that it guarantees that all compatible producer and consumer summaries are found. An interesting property of the obtained CBT structure is that its stretch is sub-linear with respect to the number of nodes involved in the structure. This means that the distance between nodes in the CBT grows slowly with the size of the CBT itself, thus enabling fast operation of the CBT and the appropriate matching algorithms. It is also important to note the good scalability of the CBT in terms of the required number of core motions. This because each core motion includes the transfer of the service descriptions, which needs to be kept at a minimum. There are some drawbacks to the CBT scheme which need consideration when the cost and benefits of CBT deployment are estimated. First and foremost, the CBT requires an addressing scheme which is different to what is usually used on the Internet. However, this is necessary as the internet type addresses do not work in a MANET. Second, it requires knowledge of, and manipulation with the service descriptions, restricting the applicability of the CBT to use cases where service descriptions can be cast into the appropriate dataspace form. While this is a concern in the general case, for well defined tasks (e.g. those described by CPNs themselves) the service descriptions are known well enough.

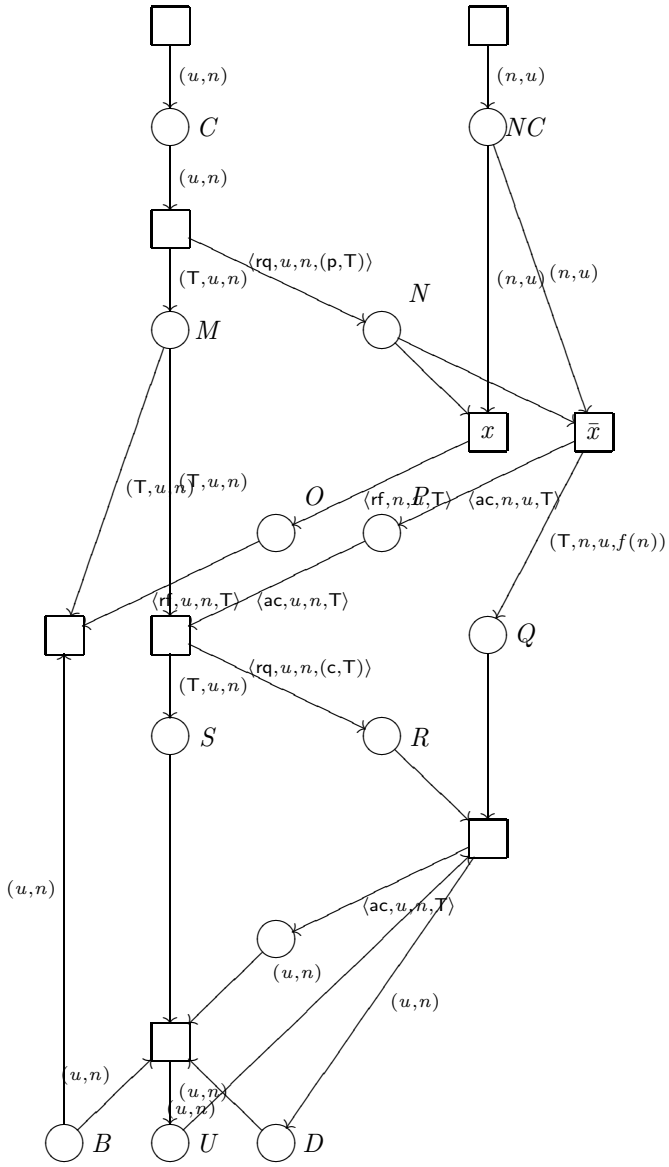


Figure 6.9: Bottom half of the Switch protocol.

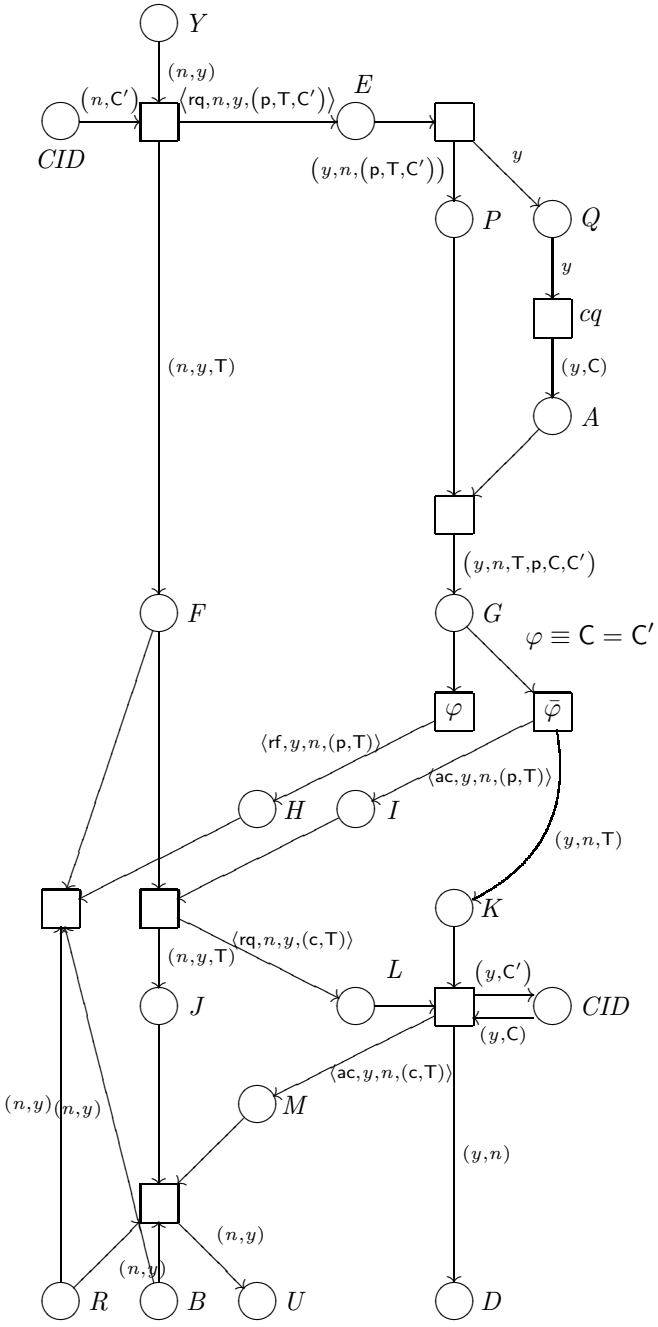


Figure 6.10: The *Yield* protocol.

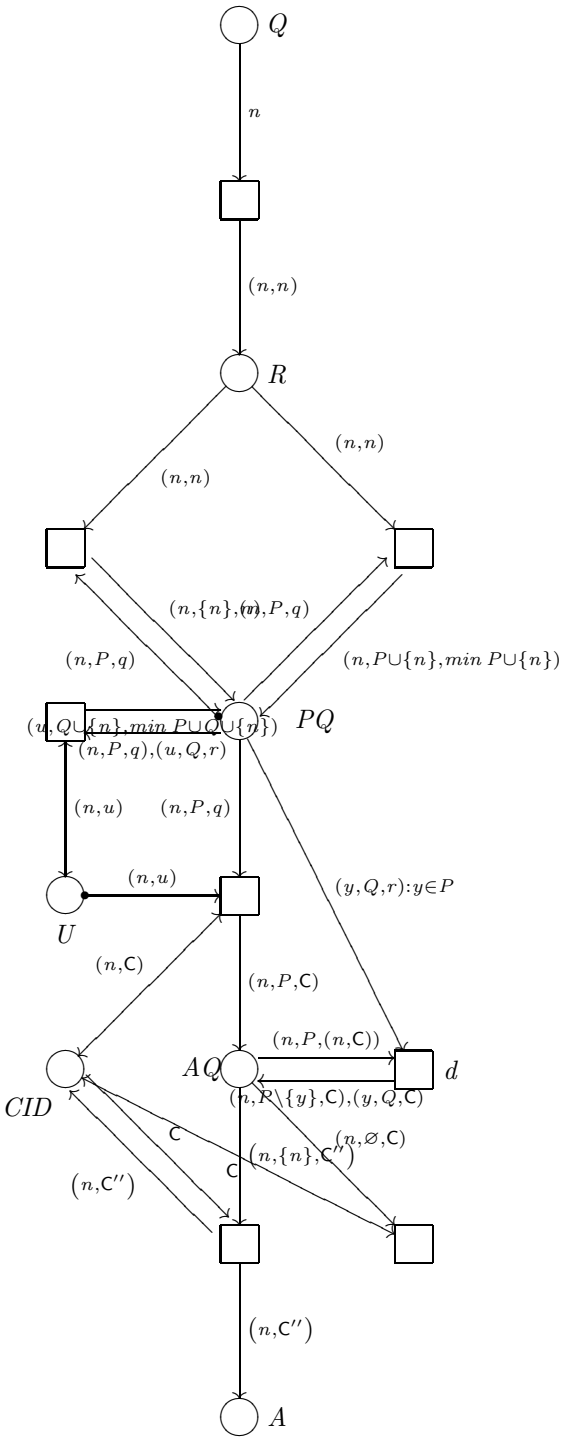


Figure 6.11: The *Yield* protocol component query.

Chapter 7

The Execution Model

In this Chapter, we lay out the model for executing distributed programs within the Distributed Workflow Execution Architecture for Mobile (DWEAM) context. The programs are described by workflows represented with CPNs. In the DWEAM context, the execution of distributed programs reduces to ensuring that the progress property holds in face of the node and network *volatility*. Provided that the progress property holds, additional safety requirements must be fulfilled so that the semantics of the program execution is equivalent to that given by its workflow description, under arbitrary workflow element instantiation.

The workflow elements, i.e. the *places* and *transitions* are instantiated on multiple nodes. There is no restriction on the instatiation cardinality for the workflow elements. Any token on a given place can be instantiated multiple times at disparate nodes. Transitions can be similarly instantiated on multiple nodes.

7.1 Introduction

The foundation of DWEAM's execution model is in the distinction between the description of a program (hence: a workflow) and its instantiation (hence: the allocation of procedures to computing elements and the location of objects representing tokens). This separation may appear a self-evident fact that buys us nothing when we want to understand distributed systems. However, the distinction is in fact important. It is disregarded more often than not, and this becomes obvious if we take a look at the contemporary distributed programs and how the missing distinction hinders the distributed applications.

Looking at the client-server model for communication, which is ubiquitous in todays Internet, we can observe that the above statement is indeed true. A prime example is given by the common scenario where a user is requesting a web page from a remote server, through the web browser client. We can consider the contents of a web page as being some sort of a remote object. The fact that the web page is usually delivered in a markup language such as Hypertext Markup

Language (HTML) is not relevant as it has to do only about the *representation* of the object offered in response to a query. The representation itself can even be the left to the client's preference. Regardless, the method used to locate the object is standardized. It is through the URL addressing scheme, which uniquely identifies every object accessible from the Internet.

Example 12 *Any user of the Internet can request the object whose content is given by the URL* `http://cobalt.et.tudelft.nl/~filip/index.html`. *In response to the request, the user is delivered the contents of the author's home page.*

An inherent part of the URL is the Fully Qualified Domain Name (FQDN) of the computer `cobalt.et.tudelft.nl`, the computer at whose disks the home page physically resides. If for some reason `cobalt` is inaccessible, so is the home page, along with any other objects `cobalt` hosts. We can conclude that the existence of the home page *object* depends on the instantiation at the particular *host*, as far as the rest of the Internet is concerned.

If the data objects were decoupled from their instantiations, such an outage would never have to occur. Unfortunately, since the host-object coupling was disregarded in Example 12, the practice dictates we are forced to worry about the well-being of a particular machine, so that the object remains accessible. Ideally, the object exists in a data space which is detached from the embodiment and the user's query for the object should retrieve it regardless of the state the embodiments of the object assume. In thinking about computer programs, we should worry about the data and the procedures, and not about the machine itself. This point was well conveyed by H. Abelson et al. [7], who teach this distinction in the course MIT CS 6.001, "Structure and Interpretation of Computer Programs". They state that computer science is a misnomer as it is neither a science (for computer science is about procedural knowledge while science deals with understanding nature through observation) nor it is about computers (in the same sense as geometry is about declarative knowledge[1] and not about rulers and compasses). It follows that in a distributed system, we should ideally not care where the actual objects we operate on are located. This is not always easy to achieve, as distributed program components can sometimes follow conflicting execution paths that contend for data. The practical goal of DWEAM is to obtain a distributed environment for program execution that takes the decoupling between programs and computers as far as reality permits.

7.2 Data Model

The goal of DWEAM is to provide a workflow execution engine that is independent of the node network location. That includes processing that does not

[1] The *declarative* knowledge is the knowledge about what is true. As opposed to the *procedural*, or *how-to* knowledge, which talks about the actions needed to bring some result about.

fundamentally depend on where the tokens are effectively stored. DWEAM can be thought of as a workflow processing machine designed to operate over wide area networks of Internet scale without regard of the physical node placement. For such a machine we need to look into two mecnahisms:

1. Location-independent data object storage; and

2. Location-independent data object processing.

These mechanisms must be conceived in a way that fully and correctly implements the workflow specification given by the system designer. The location-independent mechanisms are in contrast to today's practice on the Internet, where location-dependent mechanisms such as the URL are almost exclusively used.

A Case for Location-Independent Mechanisms

A significant complication that surfaces in distributed systems is that apart from solving computational problems, the distribution management becomes an issue. In the absence of automated mechanisms to handle the distribution issues, this concern comprises a significant portion of the program code. It is then natural to reduce this complexity by factoring the distribution management out of the main code body and promote it into a separate distribution-handling system component. This step is as important as it is complicated, because it seems difficult to do this factorization so that the connection between the distribution mechanism and the application is broken completely. This observation relates to the *model uniformity* problem as given by Homburg et al. in [41]. Talking about the communication primitives that make basic distribution possible, they observe that "[Remote Procedure Call (RPC) and other remote call mechanisms] do not provide solutions to problems that are common to all wide-area systems, such as replication, data migration and concurrency control. The effect is that each application has to invent an ad-hoc solution."

At this point it becomes apparent that the factorization cannot work unconditionally, without introducing assumptions about the method used for the program distribution. One way to handle the distribution issues is location-independent addressing. It allows one to address a particular resource, typically a data object, through an uniquely assigned handle. The unique handle is typically sampled from a large discrete space.

Workflow Design

In the Chapter 2 we presented a toolkit that is used to specify the tasks to be performed. As far as the workflow designer is concerned, the distributed execution of the worflow is further left entirely to the workflow management.

Specifically, the task of the workflow management is to instantiate enough "worker" nodes to execute the workflow, and to manage the coordination between

the workers so that the result of the instantiated workflow execution corresponds to that intended by the workflow specification. For a fully distributed system, and especially such that aspires to not rely on pre-existing infrastructure, the control of the task distribution must be distributed itself. It cannot be assumed that the control process is a centralized entity which is somehow exempt from the volatility of the entire platform. The control itself is implemented in the same volatile platform as is the instantiated workflow.

We will now review the requisites for content-based addressing which is the only addressing mode in DWEAM.

Content-based addressing is a way to refer to a data object by specifying a set of conditions that its contents must fulfill. As objects keep internal state, the conditions have to do with the internal object state. The object's access control (embodied in the access modifiers) can be used to control which parts of the object state are accessible for content-based addressing.

Content-based object selection is common practice in various forms. One form is *database querying* whereby database tables are searched for records with contents fulfilling some particular constraint. Various languages exist for specifying the data semantic models. Van der Wolf [84] used one of the available such languages, called Object Type Oriented Data (OTO-D) [5] for the data modeling.

Example 13 (Content Based Query in OTO-D) *Selecting a design component, from [84], page 77:*

> *GET ToolRun ITS Used_Opts, Designer, Start_Date WHERE Tool ITS ToolName = 'Dracula' AND Start_Date ≥ 921027*

In the Example 13, in effect a content-based address is formed from the query, and expressed in a semantic database specification language called OTO-D[2]. The address consists of the composition of two operations: the *selection*, whereby only the objects fulfilling a particular conditions are considered, and *projection* whereby only the relevant fields are selected from all the available ones in the multi-field objects. In the example, the address considers the objects of class *ToolRun*, projecting them to their three attributes, *Used_Opts*, *Designer*, and *Start_Date*, but only such that the *Tool* attribute equals to a character string "Dracula" (where *Dracula* is the name of a toolchain component), and the start date greater than "921027".

One must note that the filed matching in the given examples is purely on the syntactic level. The semantics of the object is defined externally by the data model designer. It is the designer who specifies that the string "ToolName" must contain the sequence of character corresponding to a particular executable file name, of a program used to manipulate the object. To peruse the object in a

[2]The OTO-D would in the modern times probably be replaced by some Extensible Markup Language (XML)-based suite of a document type definition and a XML parser, or by a graphical language such as the UML.

consistent way across computer systems, the semantics of the object must be known on a system-wide level. It is to say that a convention, or a standard is available, which sets the meaning of all the used objects. While it is true that due to the object-oriented approach parts of the object semantics can be captured in the class methods, using the methods the right way is still an external task, thus ensuring that the object (and hence its internal state) fits meaningfully into the broader system context. System-wide standardization of object semantics seems so far to be the only practical way to achieve such universal semantics adoption. While work is ongoing to make similar semantic models mappable to one another, the absence of widely acclaimed solutions for this problem in today's Internet-at-large, and the prevalence of single-standard solutions, e.g. those by the World Wide Web Consortium (W3C), indicate that standardisation is at least a near, to midterm, way to go. On the other hand, strict standardization and fixing the object semantics is likely too restrictive for a distributed system. Mandating a fixed set of contexts in which a particular object can be referenced goes against the idea of a heterogeneous system, where supposedly different and non-foreseen object uses may (and should) appear.

Further, allowing external-context dependent object manipulation restricts the use of class methods. Given the context dependence they can only be expected to change or export the objects context-invariant properties. Hence, we can allow the objects only to be able to export their internal state (such methods are known as *getters*), or import internal state (*setters*, analogously). By this convention we effectively dropped all class methods, except for the getters and setters themselves. No expressive power is lost however in comparison to designs that freely uses class methods for object-specific tasks. This is because such a task could encode only a single object semantics, which we already argued is not the appropriate way to handle objects in a distributed system. The decision would not be justified in the case where the object semantics was fixed on a system-wide level. In the case of system-wide agreement, the object manipulation code could be embedded within the object itself in line with the usual Object Oriented (OO) practices, hence eliminating the need for context-sensitive handling. This gives rise to an object models in which methods to manipulate objects exist separately from the object state, and resembles the object models used in languages such as Scheme or Lisp.

The *getter and setter* object methods give a way to examine object state and form queries. The approach used to examine the components of the object state (e.g. what variables are there) is called *reflection*, and is elaborated on in the following section.

Reflection

In this section we will examine the techniques that are used to inspect the contents of the objects used to transfer messages between the nodes in DWEAM. These techniques are collectively known as *reflection* and as such appear for instance

in the mainstream programming languages such as Java and Python. The Java programming language provides a facility named *Reflect* that provides reflection services. Python makes a dictionary available for each object created, containing the detail account of the object internals.

Reflection allows a program routine to inspect the contents of an object by requesting the values stored within the fields of the object, or to obtain the information about the methods that the object class supports as well as their declaration. This allows a program to reflect on its own structure, hence the facility name. The accessible fields are defined in the appropriate object's class definition, and are also accessible to the program at runtime. The access methods exist that allow the program to inspect object contents. This account is valid for dynamically typed languages like Java. It also means that, whenever a new class is created, the reflection capabilities become available by default through the reflection mechanism. For statically typed languages like C++, a similar effect is not achievable with the help of the runtime library alone. Special support from toolkits is needed that does not form a part of the C++ standard library. However, there are programmatic ways by which these features can be emulated where appropriate. It is of course convenient if the language itself provides such a facility. For this reason we confine our exposition to the environments that support reflection natively, with an understanding that extending the reflection to other environments can imply some extra considerations.

From here onwards we also assume reflection is available throughout the nodes that participate in the DWEAM system. Reflection provides us with an important operator on data objects. It is used to obtain the fields which some object consists of. In terms of Object-Z, reflection maps an object into a schema by which the class of that object is defined. Once obtained, the bindings from the class definition can be extracted, allowing the object's contents to be inspected.

Example 14 (Reflection) *Consider a schema defining a class to contain a temperature reading for some particular place:*

$$
\begin{array}{|l}
\hline
\textit{Temperature} \\
\textit{id} : \mathbb{N} \\
\textit{place} : \textit{seq}\, \textit{CHAR} \\
\textit{value} : \mathbb{R} \\
\textit{unit} : \textit{seq}\, \textit{CHAR} \\
\hline
\end{array}
$$

and a variable of that class.

$$
\begin{array}{|l}
t : \textit{Temperature}
\end{array}
$$

The type of the schema Temperature is $\langle\!|\ \textit{id} : \mathbb{N};\ \textit{place} : \textit{seq}\, \textit{CHAR};\ \textit{value} : \mathbb{R};\ \textit{unit} : \textit{seq}\, \textit{CHAR}\ |\!\rangle$ *([78], page 26). The elements of the schema Temperature*

are bindings *of the form:* $\langle id \Rrightarrow 1, place \Rrightarrow "kitchen", value \Rrightarrow 27, unit \Rrightarrow "C" \rangle$, *hence the following assignment is valid:*

$$t = \langle id \Rrightarrow 1, place \Rrightarrow "kitchen", value \Rrightarrow 27, unit \Rrightarrow "C" \rangle$$

At the same time, the contents of the binding can be reflected *as a sequence of key-value pairs as follows:*

$\{\langle name \Rrightarrow$ *"*$\mathtt{id}$*"*$, type \Rrightarrow$ *"*$\mathbb{N}$*"*$\rangle,$
$\langle name \Rrightarrow$ *"*$\mathtt{place}$*"*$, type \Rrightarrow$ *"seq* $CHAR$*"*$\rangle,$
$\langle name \Rrightarrow$ *"*$\mathtt{value}$*"*$, type \Rrightarrow$ *"*$\mathbb{R}$*"*$\rangle,$
$\langle name \Rrightarrow$ *"*$\mathtt{unit}$*"*$, type \Rrightarrow$ *"seq* $CHAR$*"*$\rangle\}$

Dataspace

In the previous chapters, the notion of *Dataspace* was introduced as an opaque type. Operations on this opaque type were introduced too. At this point we will unpack this opaque type and expose its internal structure. We equip it with concrete content to represent in detail what it means. This is because we will be needing operations and predicates talking about parts of the *Dataspace*. Informally, the *Dataspace* is a supertype that admits all the instances of all variables having all possible constructible types. This is of course not a definition in the strict sense because it gives no clue how "all possible" values of "all possible types" can be obtained. It does not give a clue whether a type with these properties at all exists. We will therefore give a constructive way to build the *Dataspace* from the suitably chosen *primitive* types and set of chosen *constructors* that combine these types into more complex constructs. Such an approach allows us to build the *Dataspace* inductively.

The first step in constructing is the establishment of the *basic types*. Here, the basic types are considered within the context of the computer system for which the *Dataspace* is implemented. We adopt somewhat platform-dependent but yet common types for this purpose. The argument in favor of the choice is that all practical computer platforms have such basic types in one form or other. These types draw their properties from the adopted machine architecture. For our purpose the exact choice of the basic types is not essential. But we follow the common practice and adopt the integers ($\mathbb{Z}$), the reals ($\mathbb{R}$, represented by a floating point approximation) and the strings (seq $CHAR$).

$$Atoms \ ::= \ Naturals\langle\langle\mathbb{Z}\rangle\rangle \mid Reals\langle\langle\mathbb{R}\rangle\rangle \mid Strings\langle\langle \text{seq}\, CHAR\rangle\rangle$$

We further define *constructors* used first to compose aggregate objects from the basic ones. The constructors can be used to aggregate the newly obtained objects into more complex aggregates. Constructors we consider are *sets* (i.e. set

types), *sequences* and new *schemes*.

$$
\begin{array}{l}
[\ \mathit{SchemaDefs}\ ] \\
\mathit{Compounds} \ ::= \ \mathit{Sets}\langle\langle \mathbb{P}\ \mathit{Compounds} \cup \mathit{Atoms} \rangle\rangle \\
\quad \mid \mathit{Sequences}\langle\langle \mathrm{seq}\ \mathit{Compounds} \cup \mathit{Atoms} \rangle\rangle \\
\quad \mid \mathit{Schemes}\langle\langle \mathit{SchemaDefs} \rangle\rangle
\end{array}
$$

The *SchemaDefs* represents the aggregate schema types. These are either schemas with a single element, or schemas obtained by attaching a single schema element to a previously existing schema. The single schema has a generic type identifier *id*, which can be specified at construction time, corresponding to the free choice for the component name. As the only manner to do so we have, for simplicity, chosen aggregation, i.e.. packing an arbitratry number of state variables into a single schema. Every schema with $n > 1$ elements can be recursively decomposed into a schema containing $n - 1$ elements and a schema containing precisely a single element. We don't consider empty schemas specially.

$$
\begin{array}{l}
\mathit{SchemaDefs} \ ::= \ \mathit{SingleSchema}\langle\langle\ \{\!|\ \mathit{id} : \mathit{Dataspace}\ |\!\}[\mathit{id}] \rangle\rangle \\
\quad \mid \mathit{MultiSchema}\langle\langle\ \{\!|\ \mathit{id} : \mathit{Dataspace}\ |\!\}[\mathit{id}] \times \mathit{SchemaDefs} \rangle\rangle
\end{array}
$$

Finally, the *Dataspace* is obtained from the three above components, completing the specification of the *Dataspace*.

$$\mathit{Dataspace} \ ::= \ \mathit{Atoms} \mid \mathit{Compounds} \mid \mathit{SchemaDefs}.$$

Cubes and Cube Sets

An important functionality based on *Dataspace* is the ability to focus on a particular subset. For this we consider the *selection* operator. It alows us to specify a subset of the *Dataspace* for which we have a particular interest in, as it provides the ground upon which the matching algorithm is built later in the text.

By analogy with physical space, each valid binding (i.e. variable instance) for the type *Dataspace* we call a *point*. A point of the *Dataspace* is uniquely determined by a binding that assigns an unique value to each component in the corresponding type schema.

Example 15 (Compound Schema) *With reference to the Temperature schema of Example 14, we can define another schema that contains it.*

Reading

$t : \mathit{Temperature}$

$\mathit{humidity} : \mathbb{R}$

This schema is constructed from a single basic type, aggregated with a component of a previously defined compound type.

A point in the Dataspace is then determined by a binding that assigns a unique value to both the components of the schema. The bindings can also be nested, as is used in this case:

$$\langle t \Rrightarrow \langle id \Rrightarrow 1, place \Rrightarrow "kitchen", value \Rrightarrow 27, unit \Rrightarrow "C" \rangle, humidity \Rrightarrow 0.6 \rangle.$$

For the data distribution we need to address larger subsets of the *Dataspace* than the individual points. These subsets are then used as objects for common operations, such as the service discovery. The subsets are obtained by the *selection* and *projection* operations which constrain the values that can be taken on by the individual *Dataspace* schema components. For this purpose we consider each schema component to be a *coordinate*. A coordinate is a reflection of a schema component available in the code. We introduce the reflection type specifier *Type*. This type specifier contains the necessary data to uniquely identify the type it refers to. It is for instance enough that it contains a string representation of the type name.

$$[\ Type\ ]$$

On the other hand, a coordinate is determined by both the variable type and the name used to identify the respective schema component. Such a designation is needed because multiple schema components can be of the same type. Therefore, type alone does not suffice to identify the component, whereas the pair of name and type does.

$$Name == \mathrm{seq}_1\ CHAR$$

Hence, a coordinate is given as:

Coordinate

$$name : Name$$
$$type : Type$$

At this point it is necessary to introduce the **typeof_** and **reflect_** *pseudo-operators*. These pseudo-operators are difficult to express in the Z notation, as it does not support self-referencing. Hence we use the name pseudo-operators, so as to emphasize the divergence from the rules of Z. The two operators will therefore be introduced through an example, and an informal explanation of their operation. This oddity is not problematic in practice however, for as long as the reflection is in the vocabulary of the used implementation language. As such languages routinely exist (Java being one of them) we consider this an acceptable deviation from the Z formalism.

Example 16 (Applying reflection) ***typeof_*** *is a pseudo-operator which, for a given instance variable of a particular type, returns as a result the actual type*

from which t takes its value. ***reflect_*** *is a pseudo-operator which, given a type returns a set of bindings to Coordinate objects, with textual representations of the given schema elements.*

Consider the schema Temperature as given in the Example 14, and the variable t defined to have this type. Then:

$$\begin{aligned}
&\textbf{\textit{typeof}}\, t = \mathit{Temperature}\\
&\textbf{\textit{reflect}}\,(\textbf{\textit{typeof}}\, t) =\\
&\quad \{\langle \mathit{name} \Rrightarrow \text{“}\texttt{id}\text{”}, \mathit{type} \Rrightarrow \text{“}\mathbb{N}\text{”}\rangle,\\
&\quad \langle \mathit{name} \Rrightarrow \text{“}\texttt{place}\text{”}, \mathit{type} \Rrightarrow \text{“}\mathit{seq}\, \mathit{CHAR}\text{”}\rangle,\\
&\quad \langle \mathit{name} \Rrightarrow \text{“}\texttt{value}\text{”}, \mathit{type} \Rrightarrow \text{“}\mathbb{R}\text{”}\rangle,\\
&\quad \langle \mathit{name} \Rrightarrow \text{“}\texttt{unit}\text{”}, \mathit{type} \Rrightarrow \text{“}\mathit{seq}\, \mathit{CHAR}\text{”}\rangle\}
\end{aligned}$$

Each *Coordinate* can take on any value from its support type. If we want to consider only parts of the entire support set (denoted as X), a constraint can be specified. The constraint is generic with respect to the support set. Conceptually, for a support set X, a constraint on it is a subset of X.

$$\mathit{Constraint}[X] == \mathbb{P}\, X$$

Encoding the constraint for a generic set X requires the description of all the elements that form the constraint. However, for sets that have a known internal structure, encoding the constraint can be more efficient and be significantly accelerated. For *totally ordered* sets, for instance, specifying a continuous interval amounts to the naming of the bounds, which can be encoded to yield a description of constant size, provided that the description of each single element from the set is constant size itself. As this is often the case in computer systems, where practically all elementary data types are either of constant size (e.g. integers and floating point numbers), the constant size encoding assumption is reasonable. Partially ordered sets exibit a similar structure, with the addition that the description size depends on the ordering relation. In such cases, multiple elements may have to be counted as comprising the bounds.

Example 17 (Constraints) *Integer and floating-point intervals, as represented in finite precision in a computer's memory, can be encoded in constant space by specifying the upper and lower bounds only with respect to the ordering relation* $\_ \leq \_$ *, in interval notation as:* $[3, 4]$*, or* $[2.71, 3.14]$*.*

String intervals (seq CHAR) can be encoded in a similar way, with the respect of the lexicographic ordering. In interval notation, an example thereof is [*“aardvaark”*, *“zebra”*]*.*

Specifying a constraint for some coordinate yields a *surface*. Multiple surfaces on the components of the same type X taken together, comprise a *cube*. The cube is a part of the *Dataspace* where some coordinates are constrained to intervals. These are the basic *Dataspace* subsets that can be operated on. Set operations on

cubes should be supported. These include testing whether a point is a member of a given cube, as well as union, intersection and complementation of particular cubes.

$$Surface == Coordinate \times Constraint$$
$$Cube == \mathbb{P}\, Surface$$
$$CompoundCube == \mathbb{P}\, Cube$$

The cube membership of a *Dataspace* point is defined through the use of the reflection operators.

$$\_ \in \_ : Dataspace \leftrightarrow Cube$$
$$\forall\, x : Coordinate \in first \bullet \forall\, y : Cube \bullet first(y) \in \mathrm{dom}\, \mathbf{reflect\ typeof}\,(x) \wedge \mathbf{valueof}\,(x, first(y)) = second(y) \Rightarrow (x, y) \in (\_ \in \_)$$

A set of cubes that result from these operations are *compound cubes* that should be also adequately supported with the set operators.

$$\_ \in \_ : Dataspace \leftrightarrow CompoundCube$$
$$\forall\, x : Dataspace, c : CompoundCube \bullet \exists\, y : Cube \bullet y \in c \wedge x \in y \Rightarrow (x, c) \in (\_ \in \_)$$

With the overloaded definition of the relation $\_ \in \_$, all the set operations for the new types *Dataspace*, *Cube* and *CompoundCube* are defined. Their specifications are therefore omitted.

For each type X whose instances are points in the *Dataspace*, the *empty* and *full* cubes are defined as cubes that, respectively, for each coordinate, have no elements in the constraint (denoted as $\square[X]$); and extend over the entire domain (denoted as $\blacksquare[X]$).

$$[X]$$
$$\square[X] : Cube[X]$$
$$\blacksquare[X] : Cube[X]$$
$$\forall\, c : Surface \bullet first(c) \in \mathbf{reflect}\ X \wedge c \in \square[X] \Rightarrow second(c) = \varnothing$$
$$\forall\, c : Surface \bullet first(c) \in \mathbf{reflect}\ X \wedge c \in \blacksquare[X] \Rightarrow second(c) = X$$

Following the usual Z conventions, the generic type argument (X in the above generic schema) can be omitted if the type X can be inferred from the context in which it is used.

Example 18 (Type Inference) *Let a variable t be defined as:*

$$t : Temperature$$

where Temperature is the type introduced in the Example 14. Then, the expression:

$$t \in \blacksquare$$

is equivalent to:

$$t \in \blacksquare[Temperature]$$

because $\textbf{\textit{typeof}}(t) \equiv Temperature$ *from the above definition.*

Finally, for convenience, to denote the set union (usually $\_ \cup \_$) and set intersection operators (usually $\_ \cap \_$), we use $\_ + \_$ and $\_ \cdot \_$ as they make for clearer notation, especially where the operator symbol can be dropped, as is the case in the expression $a \cdot b = ab$.

$$\_ + \_ == \_ \cup \_$$
$$\_ \cdot \_ == \_ \cap \_$$

Operations on the Cubes and Cube Sets

Conceptually, once the element containment relation ($\_ \in \_$) is overloaded for the *Dataspace* and the cubes, all the usual set operations are well defined. However, this does not take into account how practical these operations are from a computational point of view. Namely, depending on the order in which the operations are performed, the size of the compound cubes describing a particular set can differ, in terms of the number of elements that the respective set has. This translates directly into a varying amount of space used to represent the compound cube in the program runtime environment. Further, more complex descriptions make for more processing time spent for obtaining a result of a typical set operation. It therefore seems evident that smaller size descriptions are preferred over larger ones as compact descriptions save both the processing time and space.

Intersection

When specifying a method to compute the set operations on cubes and cube sets, it is convenient to express these in terms of cubes that themselves do not overlap. Non overlapping cubes allow for the simpler manipulation when determining the size of the compound cube sets. In other situations, one may opt for the simplest

description of the cubes to reduce the required memory footprint[3]. The simplest way to achieve this is with the intersection of individual *Cube* objects. The intersection is defined as a binary operator, taking two arguments, both of the type *Cube*.

Let these be $x \in Cube$ and $y \in Cube$. The cube intersection computation can be factored into intersection computation along each of the coordinates. Only the coordinates with constraints that are shared between x and y are intersected. The intersection of the two is placed into the resulting cube. The coordinates which are not shared are copied from the respective schemas that contain it, into the resulting schema. Non-shared coordinate constraints (i.e. a coordinate which is constrained in one cube only and not in the other) are copied into the resulting cube. This is because the coordinates that are not mentioned in the constraint are assumed unconstrained, and the unconstrained coordinate is the identity for the intersection operator.

$$
\begin{array}{|l}
\_ \cdot \_ : Cube \times Cube \to Cube \\
\hline
\forall\, x, y, z : Cube \bullet \\
\quad (\textbf{let}\ onlyx == \{c : Coordinate \mid c \in \operatorname{dom} x \land c \notin \operatorname{dom} y\}; \\
\qquad onlyy == \{c : Coordinate \mid c \notin \operatorname{dom} x \land c \in \operatorname{dom} y\}; \\
\qquad common == \{c : Coordinate \mid c \in \operatorname{dom} x \land c \in \operatorname{dom} y\} \bullet \\
\qquad\quad z = (onlyx \lhd x) \cup (onlyy \lhd y) \\
\qquad\qquad \cup \{(k, c) : Surface \mid k \in common \land c = x(k) \cdot y(k)\}) \\
\qquad\qquad\quad \Leftrightarrow ((x, y), z) \in (\_ \cdot \_)
\end{array}
$$

The intersection of a cube c and a cube set cs is naturally defined by the set containing the intersection of c with all the cubes in cs.

$$
\begin{array}{|l}
\_ \cdot \_ : Cube \times CubeSet \to CubeSet \\
\hline
\forall\, c : Cube, cs, r : CubeSet \bullet \\
\quad r = \{q : Cube \mid \exists\, s : Cube \bullet s \in r \land q = s \cdot c\} \\
\qquad \Leftrightarrow (c, cs) \mapsto r \in (\_ \cdot \_)
\end{array}
$$

Finally, two cube sets are intersected by computing element-wise intersections. These are easily expressible in terms of the previously overloaded operators.

$$
\begin{array}{|l}
\_ \cdot \_ : CubeSet \times CubeSet \to CubeSet \\
\hline
\forall\, x, y, z : CubeSet \bullet \\
\quad z = \bigcup_{i \in x} x \cdot y \Leftrightarrow (x, y) \mapsto z \in (\_ \cdot \_)
\end{array}
$$

[3]A situation is analogous to that between the uncompressed data stream, which is easy to process but potentially takes up a lot of storage, and the compressed data stream, which takes little storage but cannot be readily operated on.

Union

Union computation is slightly more involved than the intersection computation, as each union operation yields a cube set. Moreover, extra care needs to be taken that the resulting cube set is disjoint. A union of two cubes x and y, contains three sets, namely $x \backslash y$, $x \cdot y$ and $y \backslash x$. The intersection can be computed using the definitions given before, whereas computing $\_\backslash\_$ deserves extra care.

Efficient difference computation has been constructed following the definition of the *sharp* operator described by de Micheli (Section 7.3 in [23]) and invented by Brayton et al [15], and further in Watanabe et al [86]. De Micheli explains the difference computation in terms of the sharp operator for *positional cube notation* and strictly binary coordinates. We extend this definition to general types, i.e. types that can be constructed as subtypes of the *Dataspace*, and that can be reflected into a set of surfaces. We first present the sharp operators as given by de Micheli, and then extend them to handle generalized types.

The cubes of de Micheli are represented in the *positional cube notation*. This is a binary encoding of the constraint, whereby an k-element set is encoded by a binary string literal of length k. A "1" at the position i of the k-element string denotes that the element i is included in the constraint. Conversely, a "0" means that the element number i is excluded from the constraint.

Example 19 (Positional cube notation) *Let the elements of the support set be* 0, 1, *and* 2. *The constraint, expressed in the positional cube notation will therefore be a 3-digit binary string literal. The string* 001 *encodes a constraint for which only the element* 2 *is included. The string* 110 *encodes a constraint for which elements* 0 *and* 1 *are included.*

Cubes are encoded as collections of positional cube denoted constraints. As the number of coordinates in the de Micheli case is fixed, any cube can be encoded by fixing the order in which the positional cube denoted literals appear, and fixing the order in which the support set encodings appear.

Example 20 (Cube encoding) *Let there be defined three coordinates, respectivelly with the three support sets in the order as follows:* $\{0, 1, 2\}$, $\{a, b, c, d\}$ *and* $\{\alpha, \beta\}$.

A valid encoding of a cube with elements from the set $\{0, 1, 2\} \times \{a, b, c, d\} \times \{\alpha, \beta\}$*: is:*

011 1101 11,

denoting a cube formed by:

$$\{1, 2\} \times \{a, b, d\} \times \{\alpha, \beta\}.$$

In the positional cube encoding, each coordinate can be complemented easily, by making a bitwise complement of the corresponding positional cube encoding.

For the positional cube encoding 1101 in the Example 20, the complement in the positional cube notation is 0010. There are two ways to obtain the difference between two cubes. The first is by using the *sharp* operator. Applying the sharp operator to two cubes a and b, described by their coordinates: $a_1\ a_2\ \ldots\ a_n$, and $b_1\ b_2\ \ldots\ b_n$ is given as:

$$a \# b = \begin{cases} a_1 \cdot b_1' & a_2 & \ldots & a_n \\ a_1 & a_1 \cdot b_2' & \ldots & a_n \\ \ldots & \ldots & \ldots & \ldots \\ a_1 & & \ldots & a_n \cdot b_n' \end{cases} \tag{7.1}$$

where the dot-operator ($\_\cdot\_$) denotes a bitwise *and* between the corresponding coordinate bit-strings, and the prime operator ($\_'$) denotes bitwise complementation of the corresponding coordinates. The result of the operation is a *set* of cubes, with non-empty intersection. A slightly modified sharp operator, the *disjoint sharp* yields the result in terms of the cubes with empty pairwise intersection:

$$a \circledcirc b = \begin{cases} a_1 \cdot b_1' & a_2 & \ldots & a_n \\ a_1 \cdot b_1 & a_1 \cdot b_2' & \ldots & a_n \\ \ldots & \ldots & \ldots & \ldots \\ a_1 \cdot b_1 & a_2 \cdot b_2 & \ldots & a_n \cdot b_n' \end{cases} \tag{7.2}$$

All the cubes in $a \circledcirc b$ are disjoint as all the cube pairs have at least one field with an empty intersection. The field i (i.e. $a_i \cdot b_i'$) has an empty intersection with the field i of all the other cubes $j > i$ (which is $a_i \cdot b_i$).

Example 21 (Sharp operators, [23] page 292) *Let us consider the 2D space denoted by: $U = 11\,11$. Consider the cube ab with $a = 01\,01$. Let us compute the complement of ab by subtracting it from the U with the* sharp *operation:*

$$11\,11 \# 01\,01 = \begin{cases} 10\,11 \\ 11\,11 \end{cases} \tag{7.3}$$

The expression forms the complement $a' + b'$. (The operator $\_ + \_$ is used in the union *sense). Using the* disjoint sharp *operator:*

$$11\,11 \circledcirc 01\,01 = \begin{cases} 10\,11 \\ 01\,10 \end{cases} \tag{7.4}$$

and the result is now $a' + a \cdot b'$ which is a disjoint cover.

The operators we use here are derived directly from the $\_ \# \_$ and $\_ \circledcirc \_$ operators as shown in the Example 21. There exist differences in how the operators are applied in the case of *Dataspace* cubes, when compared to the positional cubes. Two issues to overcome to define the sharp operations on the *Dataspace* are:

1. The ordering of the coordinates is not predetermined; and

2. The *universal* full cube $\blacksquare[Dataspace]$ is not well-defined, hence expressions like a' where $a \in Cube$, and $\_'$ is the complementation operator, is also not well-defined.

Both issues can be dealt with efficiently, by modifying the usual sense in which the set operations are performed on the points in *Dataspace*, and provided that the resulting properties of the operations are sufficient.

As the coordinate ordering on the types from *Dataspace* is not predetermined, stemming from no imposed ordering to Z schema components, the positional notation is not adequate. Hence the coordinate names must be featured explicitly when displaying the cubes and operations. The lexicographical coordinate ordering is introduced artificially (by the function $\mathrm{lex}\_$ to be defined) so that the sharping-off schema becomes applicable. We do not go into the tradeoff of the coordinate ordering choice, however.

$$\mathrm{lex} : \mathbb{P}\, Coordinate \rightarrow \mathrm{seq}\, Coordinate$$

$$\forall x : \mathbb{P}\, Coordinate;\ y : \mathrm{seq}\, Coordinate \bullet \\ \quad \mathrm{ran}\, y = x \land \mathrm{dom}\, y = 1..\#x \Leftrightarrow x \mapsto y \in \mathrm{lex}\_$$

As the universal full cube $\blacksquare[Dataspace]$ is not well defined, the complementation of a cube is only valid within the type that the cube describes. Thus, a complement c' of a cube c representing a subset of the type $Cube[Temperature]$ (schema *Temperature* is defined in the Example 14) is also a subset of

$$Cube[Temperature],$$

such that:

$$\langle c, c' \rangle \,\mathrm{partitions}\, \blacksquare[Temperature].$$

Hence the complement also has no intersection with any $\blacksquare[X]$ when

$$X \neq Temperature.$$

This is in contrast to the case of Example 21, where a complement of a given cube is with respect to some universal cube U. Here, the complement of a cube is only defined with respect to the type the cube has been constructed for.

The definitions of the sharp and disjoint-sharp operators in our context is given by the following generic schema:

$$
\begin{array}{|l}
\hline
[X, Y] \\
\_\#\_ : Cube[X] \times Cube[Y] \to CubeSet \\
\_\circledplus\_ : Cube[X] \times Cube[Y] \to CubeSet \\
\hline
X \neq Y \bullet \\
\quad \forall\, x : Cube[X];\ y : Cube[Y] \bullet \\
\qquad \_\#\_ = \{(x, y) \mapsto \{x\}\} \\
\qquad \_\circledplus\_ = \{(x, y) \mapsto \{x\}\} \\
X = Y \bullet \\
\quad \forall\, x, y : Cube[X];\ z : Cube[X] \bullet \\
\qquad \forall\, m \in \blacksquare[X] \oplus y \bullet \\
\qquad z = \blacksquare[X] \oplus (x \oplus (first(m), x(first(m)) \cdot (X \setminus y(first(m))))) \\
\qquad\quad \Leftrightarrow (x, y) \mapsto z \in \_\#\_ \\
\qquad (letc == \mathrm{lex}(\textbf{reflect}\ X); \\
\qquad xc == \blacksquare[X] \oplus x; \\
\qquad yc == \blacksquare[X] \oplus y \bullet \\
\qquad\quad \forall\, k \in \mathrm{dom}\, c \bullet \\
\qquad\qquad (x, y) \mapsto \mathrm{ran}\langle xc(c(1)) \cdot yc(c(1)), \ldots, \\
\qquad\qquad\quad xc(c(k)) \cdot yc(c(k))', xc(c(k+1)), \ldots, \\
\qquad\qquad\quad x(c(\#c))\rangle \in \_\circledplus\_) \\
\hline
\end{array}
$$

With the sharp ($\_\#\_$) and disjoint sharp ($\_\circledplus\_$) operators defined, the the sharp operation for cube sets can be built, ultimately admitting the union computation for cube sets as well.

$$
\begin{array}{|l}
\hline
\_\circledplus\_ : CubeSet^2 \to CubeSet \\
\hline
\forall\, c_1 = \{q_1^1, q_2^1, \ldots q_{\#c_1}^1\} : CubeSet; \\
\quad c_2 = \{q_1^2, q_2^2, \ldots q_{\#c_2}^2\} : CubeSet; \\
\quad r : CubeSet \bullet \\
\qquad r = q_1^1 \circledplus q_1^2 \\
\qquad\quad \cup (q_1^1 \circledplus q_2^2) \circledplus (q_1^1 \circledplus q_1^2) \\
\qquad\quad \cup (q_1^1 \circledplus q_3^2) \circledplus (q_1^1 \circledplus q_2^2) \circledplus (q_1^1 \circledplus q_1^2) \\
\qquad\quad \vdots \\
\qquad\quad \cup (q_1^1 \circledplus q_{\#c_2}^2) \circledplus [(q_1^1 \circledplus q_{\#c_2-1}^2) \circledplus \ldots \circledplus (q_1^1 \circledplus q_1^2)] \\
\qquad\quad \cup (q_2^1 \circledplus q_1^2) \circledplus [\cup (q_1^1 \circledplus q_{\#c_2}^2) \circledplus (q_1^1 \circledplus q_{\#c_2-1}^2) \circledplus \ldots \circledplus (q_1^1 \circledplus q_1^2)] \\
\qquad\quad \vdots \\
\qquad\qquad \Leftrightarrow (c_1, c_2) \mapsto r \in \_\circledplus\_ \\
\hline
\end{array}
$$

Finally the union computation for the cube sets is concisely defined through the use of the disjoint sharp ($\_\circledplus\_$) operator and the cube set intersection operator.

$$\_ + \_ : CubeSet^2 \rightarrow CubeSet$$

$$\forall x, y, z : CubeSet \bullet$$
$$z = x \bigoplus y + x \cdot y + y \bigoplus x \Leftrightarrow (x, y) \mapsto z \in \_ + \_$$

As now the operational definition of the union and intersection operators are available, we can investigate how these are used for content-based addressing and object delivery.

Predicates and Summaries

In content-based addressing, the addresses used for the object delivery are specified in forms of predicates that select a point subset of the *Dataspace* of interest to the communication participants. Typically, the predicates can be encoded compactly, so that the resulting description is easy to store and manipulate. For compact description though, this requires some knowledge of the structure of the alphabet (i.e. objects used) involved in the communication.

In every communication there are two roles: that of a *sender* and that of a *receiver*. In the communication based on the transfer of objects between the sender and the receiver, it is customary to call the sender a *producer* (as it produces the communicated objects) and the receiver a *consumer* (as it consumes the communicated objects). Further, a communication session can have multiple participants fulfilling either of the roles. In contrast to the address based communication that is the dominant way for message transport in electronic mail, for example, where one-to-one communication mode is still dominant, the content-based communication has in principle unlimited number of slots for either role. This is of course conditioned on the ability of the producers and the consumers to locate each other in the network.

A simple example of content-based addressing is reflected in two of the oldest Internet-based services. These are the Internet Relay Chat (IRC) and the *Usenet* services. Interestingly enough, these services with inherrent many-to-many communication ability have lost in popularity with the wide acceptance of the Internet, despite their potential, in favor of centralized services such as the World-Wide Web (WWW) and Instant Messaging (IM). The content based addressing present in these services are:

1. *Data space partitioning.* In this method of group communication, the topic sets partition the entire data space. This method is employed by IRC.

2. *Data space partitioning with hierarchical refinement.* In this method, the topic sets partition the entire data space, but can themselves be subdivided and partitioned, thus creating a hierarchical topic ordering. This method is employed by Usenet.

In both IRC and Usenet, the producers and the consumers gather around common topics (reflected in the notions of IRC *channels*, and Usenet *conferences*).

All the participants can equivocally be both the producers and the consumers. In the case of IRC, the topics are funneled in a flat channel-based topic registry. In the case of the Usenet, the topics are organized in a hierarchical fashion. A participant can therefore narrow down to the topic of own interest, by making a series of subtopic filters that refine the interest until the participant decides the granularity is appropriate.

Here we investigate a natural extension to the topic filtering methods. Rather than imposing a data space partitioning like IRC or hierarchical selection rules like Usenet, we require that the participants can choose an arbitrary *Dataspace* subset. This is achieved through the use of *summaries*. The summaries allow a compact description of the *Dataspace* of interest for the conversation participants. Conceptually the summaries are no more than *CubeSet*s and can be processed as such.

$$Summary == CubeSet.$$

The reason we introduce this new concept atop the existing cube sets, is that there are two classes of summaries needed, one for each of the two roles in the communication. We therefore distinguish the *producer* and the *consumer* summaries. The summaries are essential for the *service discovery* (refer to Chapter 6). The producer and the consumer summaries that have a non-empty intersection are considered *compatible*. That is, objects originating at the respective producer must be delivered to the respective consumer. The process that solves the service discovery problem is called *matching*.

7.3 Matching

Matching is a process used to solve the Service Discovery Problem (SDP) (Chapter 6). Informally, it consists of looking at the descriptions of the products supplied by the producers, the products demanded by the consumers, and finding whether such descriptions have commonalities. By using matching on all the nodes in the system, and giving guarantees that any producer-consumer pair is catered to, allows one to verify the solution of the SDP. Matching usually involves three parties. These are the aforementioned *producers* and *consumers* (henceforth the *clients*), and an intermediate *broker*. The broker is a process that posesses enough information to discern whether particular producers and consumers are compatible. The broker process adds the necessary loose coupling between compatible clients. The broker can be implemented as a forwarding service (i.e. multiple nodes may delegate the broker role although the producers and consumers are unaware of this). Matching consists of two sub-processes:

1. *Match guarantee.* A client must locate an appropriate broker to forward the summaries to. The broker must *guarantee* that a complementing client[4]

[4]A producer is complementing for a consumer, and vice-versa.

will be found if it exists. We further postulate that the broker must find *all* the complementing clients.

2. *Match notification.* When the match is found, the matching clients must be *notified* that the match has been made.

Various strategies exist for handling the matching. There does not seem to be a universal strategy that works well in all the circumstances. There are dependencies upon the number of clients requesting assistance from the broker, the network-wise distribution of the nodes, as well as the connectivity in the network itself. We identify three classes of strategies, as follows.

1. *Centralization.* In this strategy, there is a single (physical) broker entity to which all the producers and the consumers connect.

   The upside of centralization is the simplicity. There is a single point of contact for all the clients, and there is a single place where all the relevant data is stored, and is available for any pre-processing.

   The downside of centralization is that the broker is forcibly becoming a bottleneck. Any dip in the quality of service rendered by the broker reflects immediately on the system operation. The broker outage means that all matching service is inaccessible.

2. *Forwarding.* In this strategy, there are multiple independent but cooperating brokers. Every broker accepts clients, and tries to match the clients locally. On success, the clients are notified. On failure, the clients are forwarded to another broker that may be eligible to help further. The client must than re-iterate the request with another broker.

   The upside of forwarding is its distributed nature. Provided that a client can obtain a reasonably current broker list, it becomes increasingly difficult for the outage to occur, as opposed to the centralization strategy.

   The downside of forwarding is that it requires clients to be able to connect to any broker when needed. This condition may be difficult to fulfill if the network does not exibit full connectivity (e.g. when it is proximity-based as is the case in the MANETs), and the broker can forward the client to any part of the network in principle. Further downside is that the distribution requires up-to-date forwarding rules fulfilling the match guarantee.

3. *Delegation.* This is a more elaborate forwarding strategy. Multiple independent and cooperating brokers exist as in the forwarding strategy. The broker accepts the client, tries to match it, and the client is notified if the match succeeds. If the match fails, the broker accepts itself to be a client's delegate, and re-issues the client's query itself, while the client is still pending the broker's reply.

   The upside of this approach is the independence of the full network connectivity, provided that the up-to-date forwarding rules are maintained.

The downside of the approach is that the forwarding rules are involved, and because of the limited connectivity, the number of brokers tends to be large. Ideally, some brokering functionality is present in every node with the access to the network.

In the wireless network scenario that we consider (see Chapter 5), the *delegation* is a viable strategy, given the limitations on the direct connectivity range imposed by the nature of the medium. Hence a *forwarding match strategy* must be formulated that governs the broker operation when delegating the match requests. In the next section we propose a strategy built upon the CBT distributed forwarding structure [60], described in detail in the Chapter 6. The CBT-based strategy we consider rests on the assumption that a CBT is established between the nodes, and that it is being actively maintained as the connectivity between the nodes changes.

Forwarding Strategy

The forwarding strategy we consider for the brokers is based on the distributed CBT structure. Conformant to the notation given in Chapter 6, we use ϕ_a^p and ϕ_a^c to denote producer and consumer summaries for some node a, respectively.

To simplify the notation of Algorithm 1, we use the familiar vector and matrix notation to represent in compact form the operations on a sequence of summaries. Thus Φ^p and Φ^c, both of dimension $k \times 1$ for some integer k, are vectors into which mutltiple producer, and consumer summaries are stacked, respectively. The matching matrix, obtained as the vectorized intersection $\Phi^p \cdot \Phi^{c\,T}$ of dimension $k \times k$, has a summary intersection in each its entry.

Further, the summary operator $\mathcal{S}$ appearing in the Algorithm 1 warrants explanation. It takes some summary ϕ as a parameter, and produces another summary $\mathcal{S}\phi$ which contains at least the points that are elements of ϕ. Hence, the summary operator may simply return ϕ, but as will be seen such a situation is not as interesting as the one where $\mathcal{S}$ makes approximate summaries, which hopefully have a simpler description than the original one.

The *Dataspace* in this and the sections to follow is an m-dimensional space, where all the coordinates are integers, for simplicity. The support set for the integers is denoted as M. Hence in this case, we are considering the set $M^m \subseteq$ *Dataspace*, as the upper bound.

Algorithm

Consider an arbitrary CBT. Assume as usual that the dynamics of the CBT change is slow, once formed.

Consider now some node n in the CBT. Denote its uplink node as u, and the set of downlink nodes $\{v_1, \ldots, v_k\}$ such that n is an uplink for v_i, for every $1 \leq i \leq k$. The matching algorithm for the node n is given in the algorithm 1.

Require: Node n. Uplink node u. A set of k downlink nodes $\{v_i : 1 \leq i \leq k\}$. A vector of product summaries $\Phi^p = \left[\phi_0^p \ \phi_1^p \ \cdots \ \phi_k^p\right]^T$. A vector of consume summaries $\Phi^c = \left[\phi_0^c \ \phi_1^c \ \cdots \ \phi_k^c\right]^T$. $\mathcal{S}$, the summarizing operator.
Ensure: An up-to-date matching matrix $\mathbf{M} = \Phi^p \cdot \Phi^{c\,T}$.
  if any component of Φ^p or Φ^c changed **then**
    Recompute $\mathbf{M}$.
    Compute $\phi^p \leftarrow \mathcal{S}\Phi^p$ and $\phi^c \leftarrow \mathcal{S}\Phi^c$
    Send ϕ^p and ϕ^c to u
    Send $\mathcal{S}\phi_i^p \cdot \phi_j^c$ to i for all j
    Send $\mathcal{S}\phi_j^p \cdot \phi_i^c$ to i for all j
  else if received ϕ^p from u **then**
    Compute $\mathcal{S}\phi_i^c \cdot \phi^p$ and send to i
    Compute $\mathcal{S}\phi_i^p \cdot \phi^c$ and send to i
  else {Initialization step}
    $\mathbf{M} = [\varnothing]_k$
  end if

Algorithm 1: Matching.

The algorithm collects the summaries from all downlinks. These summaries are indexed with numbers 1 to k, with summary i pertaining to the downlink $1 \leq i \leq k$. The producer and the consumer summaries are kept apart in the vectors Φ^p and Φ^c. Node n adds own summaries, ϕ_0^c and ϕ_0^p to Φ^p and Φ^c respectively and computes the matching matrix $\mathbf{M}$. An aggregate summary is made for the uplink by applying the summary operator $\mathcal{S}$. The uplink u also performs matching and delivers the match to n. If n is the core node, no uplink u exists and the corresponding actions are omitted.

Summaries

A producer p forms its summary ϕ_p^p and propagates it to its *uplink*, i.e. the node at which its only outgoing link points to. Likewise a consumer c forms its summary ϕ_c^c and propagates to its uplink. Uplinks are obliged to forward these summaries further, until the *core* node of the CBT is reached. In CBT, a unique path exists between any producer and consumer pair. Hence it is guaranteed that at least at core node, ϕ_p^p and ϕ_c^c will be present at the same time in the steady state. The path to the core on which the summaries are propagated is called the *thread*[5]. As the core may be located at any node, the expected length of a thread equals expected path length of a tree ($\mathrm{E}[L]$). As the path between the producer (p) and the consumer (c) consists of at most two threads, its mean length is $2\,\mathrm{E}[L]$. Without further considerations, the nodes closer to the core are more burdened

[5]Thread is formally defined on page 171.

with summaries than edge nodes, since they must remember summaries published by all nodes in their sub-tree.

This number can be reduced if it is observed that overlapping summaries can be *aggregated*, i.e. reduced to a summary that *covers* given summaries [17].

A summary ϕ_1 covers a summary ϕ_2 if it holds $\phi_1 \supseteq \phi_2$. For a summary sequence $\phi_1, \ldots \phi_k$, an *exact* summary is formed as $\phi = \bigcup_k \phi_k$. It can be seen that ϕ covers the given sequence. However, for exact matching, the computing cost for the matching can be prohibitive and it may become the bottleneck in the matching process. A matching heuristic performs approximate matching [91]. It trades off the computational complexity of exact matching for additional communication load for handling false positive traffic. Let there be given a summary $\phi_1 \subset M^m$. A cover ϕ_2 of ϕ_1 is *complete* if for every point $x \in M^m$, $x \in \phi_1 \Rightarrow x \in \phi_2$. A cover is *correct* if there exists no point $x \in M^m$ such that $x \in \phi_2$ and $x \notin \phi_1$. Approximate summaries are complete, but incorrect.

Supercube Approximation

As a simplest approximation, consider the *supercube* approximation. For the given summaries, the supercube is the smallest cube covering all the summaries. Let k summaries be given as $\langle \phi_1, \ldots \phi_k \rangle$ and let l_{ki} and u_{ki} be the lower and the upper bound of the coordinate $1 \leq i \leq m$ respectively. Define the summary operator $\mathcal{S}$ to yield a *supercube*. I.e. such an operator $\mathcal{S}$ takes a sequence of summaries and produces a new summary as: $\phi = \mathcal{S}(\phi_1, \ldots, \phi_k) = [min_k\, l_{1k}, max_k\, u_{1k}] \times \cdots \times [min_k\, l_{mk}, max_k\, u_{mk}]$.

A trade-off between false positive matches and description complexity exists. Supercubing introduces $\#(\phi - \bigcup_k \phi_k)$ points that cause false positive matches. It also simplifies the description of the k summaries to only one. Thus communicating the description of the summary induces k-fold less traffic per link, and k-fold less processing time at nodes in the respective thread. Consider a node n of the CBT, as in Algorithm 1, and either producer or consumer summary type without loss of generality. To reduce the description that covers ϕ_0 up to and including ϕ_k, n can consider the supercube approximation. A partition of the summary sets can be constructed, and each element of the partition approximated by a supercube. Choice of the partition depends on the amount of false positive traffic that is allowable. The false positive traffic amount for the node n can be expressed in terms of the probability distribution function over the elements of *Dataspace*.

In particular, let us consider the object traffic at n. Let us assume that the relative frequency for the arrival of each object from *Dataspace* to n is known. Denote this frequency as f:

$$f : Dataspace \rightarrow 0..1$$
$$\int_{Dataspace} f(x)\mathrm{d}x = 1$$

The objects that get delivered to n are the members of the aggregate summary produced by n from the exact summaries ϕ_0 to ϕ_k for some k, whereas only the exact $\bigcup_k \phi_k$ are actually needed. Thus all extra delivered objects are false positives. Assuming that all the false positive objects are equal size (as they are in our example), the fraction of false positives is given by: $\omega = \int_{x \in \mathcal{S}(\phi_0,\ldots,\phi_k) \setminus \bigcup_k \phi_k} f(x)\mathrm{d}x$. The node n can only estimate f, using the traffic received up to some time instant. As soon as a supercube approximation gives too high a fraction ω, n splits the set of summaries from the offending supercube in two and constructs a supercube for each, thus creating a new approximation.

Summary Operations

In this section we analyze the summary operator $\mathcal{S}$ in more detail. We have seen that the matching is possible when $\mathcal{S}$ is complete. It can also be readily checked that the sizes of the messages exchanged by the nodes in Algorithm 1 depend on the efficiency with which the producer and consumer summaries are represented. An incorrect, but complete operator $\mathcal{S}$ may incur false positives in the matching algorithm, so to eliminate false positives, a correct summary operator is needed. The problem arises with the size of the efficient and exact description of many summaries. Any complete summary operator is acceptable as long as the number of false positives is acceptable. For example, adopting a summary operator where $(\forall\, A \subseteq M^m)\ \mathcal{S}(A) = M^m$ yields a flooding communication strategy, at the expense of large false positive traffic.

Here we suggest the hierarchical summary operator represented in Figure 7.1. In the Figure, a set A (represented by a shaded circle) is approximated by disjoint subsets of M^m (solid rectangles), which form $\hat{A} = \mathcal{S}A$. The approximation is governed by a parameter ε, specifying the ceiling fraction of generated false positive traffic (i.e. the fraction of total transferred points of M^m as $\mathcal{S}$ is only complete). Approximation starts with a trivial summary operator (e.g. supercube). If the approximation yields more than fraction of ε of false positive traffic, the operator is refined by partitioning the range (dashed lines). The process is repeated recursivelly for each member of the partition until a sufficiently detailed approximation is found.

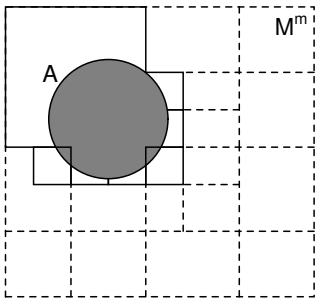


Figure 7.1: The hierarchical approximation of a set A (shaded circle).

Let us now compute the condition for the approximation refinement allowing us to find an adequate approximation to an arbitrary set of objects from *Dataspace*, at node n from the previous section. We already specified that for

simplicity *Dataspace* $\equiv M^m$. Consider now a sequence of random variables $\langle X_1, X_2, \ldots, X_l \rangle$ for some integer l, representing the object sequence received at node n. The PDF for each of these variables is equal to $f_X \equiv f$, as before. Now the false positive traffic fraction ω is given as:

$$\omega = \Pr\left((X \in A, X \notin \hat{A}) \vee (X \notin A, X \in \hat{A})\right) = \Pr\left(X \notin A, X \in \hat{A}\right) \leq \varepsilon, \quad (7.5)$$

due to the completeness of $\mathcal{S}$. Expressing $\Pr\left(X \notin A, X \in \hat{A}\right)$ in terms of the distribution density f_X:

$$\int_{x \in M^m} \left([x \in \hat{A}] - [x \in A]\right) f_X(x) \mathrm{d}x \leq \varepsilon, \quad (7.6)$$

where $[\cdot]$ denotes an indicator[6]. Now partition M^m to k approximately equal sets $M_1, \ldots M_k$ and rewrite Equation (7.6) in terms of the new sets:

$$\begin{aligned} &\int_{x \in M^m} \left([x \in \hat{A}] - [x \in A]\right) f_X(x) \mathrm{d}x = \\ &\qquad \sum_{\mu \in \{M_i : 1 \leq i \leq k\}} \Pr(\mu) \int_{x \in \mu} \left([x \in \hat{A}] - [x \in A]\right) f_{X|\mu}(x \mid \mu) \mathrm{d}x \leq \varepsilon, \end{aligned} \quad (7.7)$$

where $\Pr(\mu) = \#\mu / \#(M^m) = 1/k$, and $f_{X|\mu}$ the conditional probability of X with respect to the set μ. Finally:

$$\begin{aligned} &\frac{1}{k} \sum_{\mu \in \{M_i : 1 \leq i \leq k\}} \int_{x \in \mu} \left([x \in \mu \cdot \hat{A}] - [x \in \mu \cdot A]\right) f_{X|\mu}(x \mid \mu) \mathrm{d}x = \\ &= \frac{1}{k} \sum_{\mu \in \{M_i : 1 \leq i \leq k\}} [\mu \cdot A \neq \varnothing] \int_{x \in \mu} \left([x \in \mu \cdot \hat{A}] - [x \in \mu \cdot A]\right) f_{X|\mu}(x \mid \mu) \mathrm{d}x \leq \varepsilon, \end{aligned} \quad (7.8)$$

Each of the M_i can be further partitioned as given here, in order to obtain a more precise hierarchical description of $\hat{A}$. The hierarchical description of $\hat{A}$ hence consists of all the sets M_i and their subdivisions that have a nonempty intersection with A (hence also with $\hat{A}$). The size of the description is proportional with the number of sets in the subdivision (with the proportionality constant the size of an individual set description in the implementing code). Subtracting left-hand sides of Equations (7.6) and (7.8) we see that the two approximations differ by:

$$\frac{1}{k} \sum_{\mu \in \{M_i : 1 \leq i \leq k\}} [\mu \cdot A = \varnothing] \int_{x \in \mu} \left([x \in \mu \cdot \hat{A}] - [x \in \mu \cdot A]\right) f_{X|\mu}(x \mid \mu) \mathrm{d}x \quad (7.9)$$

[6]Indicator $[P]$, where P is some predicate has a value of 1 if the predicate is true and 0 otherwise (from [37]).

which is due to the subsets M_i that have an empty intersection with A but have been counted in the coarser approximation of Equation (7.6) and left out in the refined one (7.8).

We implemented the straightforward hierarchical summary operator as described here and in Section 7.3. The set operations operate on integer-only cubes, and the operations have been implemented as an extension of binary value only methods described in [23]. The outcome of the summary operator is shown in

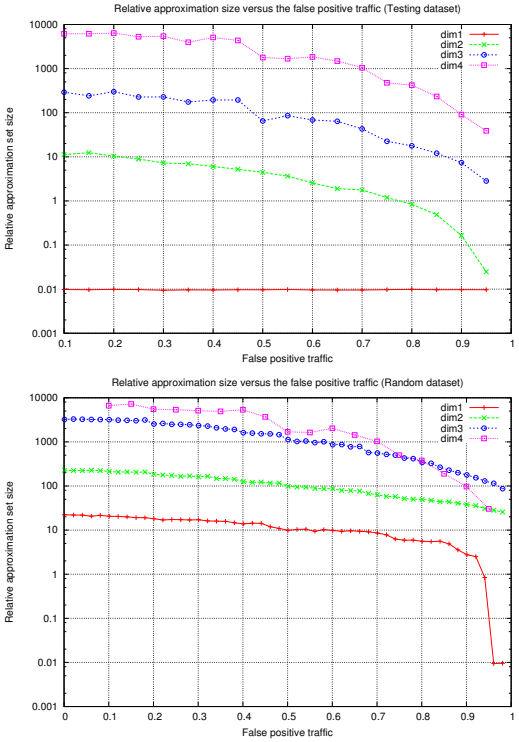


Figure 7.2: The approximation results for a range of common data cube dimensions ranging from 1 to 4.

Figure 7.2. The graphs show the relative sizes (up to a constant) of the cubes from two different data sets. The first plot is a testing data set, with intersections between cubes with dimensions from 1 to 4. The second plot shows randomly generated data sets with intersection dimensions from 1 to 4. Plots show the maximum allowed false positive traffic on the x-axis. The y-axis shows the size of the resulting data set. The unit used is the size of the runtime memory image of a single data cube[7]. Plots show a decrease in the description size with the decrease

[7]As per its current JAVA implementation. We did not optimize the size of the single data

in the summary precision.

Matching Algorithm

We now consider a matching implementation that takes the properties of the CBT into account. The basic operation is simple and follows the lines of Algorithm 1 to implement a characteristic up-down message passing, common for a wide range of algorithms executed on a tree.

Assume that a CBT has been constructed from the graph G. Let b be some node from V. Let $dn(b)$ be the set of *downlinks* of b and $up(b)$ the *uplink* of b, or $\perp$ if none. For $x \in \{p, c\}$ (p standing for *producer*, and c for *consumer*) introduce the respective summaries. Thus, ϕ_n^x is a *x-summary* of node n. Also, $\phi_{\bar{n}}^x$ is a x-summary of the subtree rooted in node n (also denote as $\bar{x}$ the complementary summary, i.e. for p, $\bar{p} = c$ and vice versa). μ_n^x is a match summary of node n. $\mu_{\bar{n}}^x$ is a match summary of the subtree rooted in the node n. $a \rightarrow b$ denotes a message sent from node a to node b. We formalize the matching requirement as follows.

Definition 11 (Matching) *Let a and b be nodes from G. Let the producer summary of a be given as ϕ_p^a, and the consumer summary of b be ϕ_c^b. A* matching *is found for a and b if there exists a node in G which determines that there exists a set $X \subseteq \phi_p^a \phi_c^b$.*

Further, the set M, such that $X \subseteq M$, must be delivered to both a and b. The matching condition determines which producers and consumers need to be connected together.

Definition 12 (Thread) *Let a be a node and there be given a set of iterated applications $up^k(a)$, for $k \geq 0$:*

$$\mathcal{T}(a) = \{a, up(a), up^2(a), \dots\} \tag{7.10}$$

$\mathcal{T}(a)$ is a thread *of a.*

Lemma 7 *The thread of any node a consists of the nodes on the path from a to the core.*

Proof. Let $\#\mathcal{T}(a)$ be the path length from a to the core node. If $\#\mathcal{T}(a) = 0$, the claim is true by definition. Assume the claim holds for $\#\mathcal{T}(a) = k - 1$. Consider a for which $\#\mathcal{T}(a) = k$. The path from a to the core node leads through upa, so $\mathcal{T}(a) = \{a\} \cup \mathcal{T}[up(a)]$. In $\mathcal{T}(a)$, a is by definition on the path of the core. Likewise, $\mathcal{T}[up(a)]$ are all on the path to the core. □

For any node b from a given CBT, the core node is a member of $\mathcal{T}(a)$. Therefore for any two nodes a and b from the CBT, there exists at least one common node in $\mathcal{T}(a)$ and $\mathcal{T}(b)$.

cube, although this is possible if needed.

Lemma 8 (Thread intersection) *Any two threads have at least one common node.*

Lemma 8 is a prerequisite for deriving the Matching condition, as it shows that there exists a node at which matching can be performed. It must be ensured that the matching occurs for compatible nodes.

Proposition 7 (The CBT Matching) *Let $\mathcal{S}$ be a complete summary operator, and let $u = up(b)$. Then the messages:*

$$b \to u: \qquad \phi^x_{\bar{b}} = \mathcal{S}[\phi^x_b + \textstyle\sum_{k \in dn(b)} \phi^x_{\bar{k}}] \tag{7.11}$$

$$\forall\, k \in dn(b)\ \ b \to k: \quad \mu^x_{\bar{k}} = \mathcal{S}[\phi^x_{\bar{k}} \cdot (\phi^{\bar{x}}_b + \mu^x_{\bar{b}} + \textstyle\sum_{l \in dn(b) \setminus k} \phi^{\bar{x}}_{\bar{l}})] \tag{7.12}$$

guarantee that matching is satisfied for all pairs of compatible producers and consumers.

Proof. The operator $\mathcal{S}$ is complete. Hence given any set $X \subseteq M^m$, it must be that $X \subseteq \mathcal{S}X$. Consider two nodes a and b ($a \neq b$) with compatible summaries ϕ^p_a and ϕ^c_b. By Lemma 8, there exists a node c such that $c \in \mathcal{T}(a) \cup \mathcal{T}(b)$. Two cases can be distinguished. First arises when $c \neq a$ and $c \neq b$. Second arises when $c = a$ or $c = b$. These cases are similar, so we work the first one out and pick the second one up when appropriate.

Case $c \neq a$ and $c \neq b$. There is a node k_a so that $k_a \in \mathcal{T}(a)$ and $up(k_a) = c$. Similarly k_b exists for b with analogous properties. We prove by induction that for any node $n \in \mathcal{T}(a)$, and any $x \in \{p, c\}$ it holds that $\phi^x_a \subseteq \phi^x_{\bar{n}}$. For $n = a$, from Equation (7.11) it holds that:

$$\phi^x_a \subseteq \phi^x_n + \sum_{k \in dn(n)} \phi^x_{\bar{k}} \subseteq \mathcal{S}[\phi^x_n + \sum_{k \in dn(n)} \phi^x_{\bar{k}}] = \phi^x_{\bar{n}}. \tag{7.13}$$

hence the base case holds. Assume now that for any sequence of up to l nodes in $\big(a, up(a), \ldots, up^{l-1}(a)\big)$ where $l < \#\mathcal{T}a$ it holds that $\phi^x_a \subseteq \phi^x_{up^{l-1}(a)}$. Using Equation (7.11), the fact that $up^{l-1}(a) \in dn[up^l(a)]$, the completeness of $\mathcal{S}$ and the inductive hypothesis:

$$\begin{aligned} \phi^x_a \subseteq \phi^x_{up^{l-1}(a)} \subseteq & \phi^x_{up^l(a)} + \sum_{k \in dn[up^l(a)]} \phi^x_{\bar{k}} \\ \subseteq & \mathcal{S}[\phi^x_{up^l(a)} + \sum_{k \in dn[up^l(a)]} \phi^x_{\bar{k}}] \\ \subseteq & \phi^x_{\overline{up^l(a)}}. \end{aligned} \tag{7.14}$$

completing the proof that $\phi^x_a \subseteq \phi^x_{\bar{n}}$ for any $n \in \mathcal{T}(a)$.

Now turn to Equation (7.12). As ϕ^p_a and ϕ^c_b are compatible, a nonempty set $X = \phi^p_a \cdot \phi^c_b$ exists. We show that X is propagated to both a and b. Consider the

nodes c and k_a. Rewriting Equation (7.12) for k_a, we get:

$$\begin{aligned} X \subseteq & \phi^p_{\bar{k}_a} \cdot \phi^c_{k_b} \\ \subseteq & \phi^p_{\bar{k}_a} \cdot \sum_{l \in dn(c) \setminus k_a} \phi^c_l \\ \subseteq & \phi^p_{\bar{k}_a} \cdot (\phi^c_c + \mu^p_{\bar{c}} + \sum_{l \in dn(c) \setminus k_a} \phi^c_l) \\ \subseteq & \mathcal{S}[\phi^p_{\bar{k}_a} \cdot (\phi^c_c + \mu^p_{\bar{c}} + \sum_{l \in dn(c) \setminus k_a} \phi^c_l)] = \mu^p_{\bar{k}_a}, \end{aligned} \tag{7.15}$$

and a similar line of reasoning shows that $X \subseteq \mu^c_{\bar{k}_b}$. Hence both k_a and k_b are notified about a match containing at least X.

It remains to show that both a and b are notified with X. The proof is also by induction, the base case thereof already given by Equation 7.15. Let l be such that $k_a = up^l(a)$, and let $k'_a = up^{l-1}(a)$. By inductive hypothesis, $X \subseteq \mu^p_{\bar{k}_a}$.

$$\begin{aligned} X \subseteq & \phi^p_{\bar{k}'_a} \cdot \mu^p_{\bar{k}_a} \\ \subseteq & \phi^p_{\bar{k}'_a} \cdot (\phi^c_{k'_a} + \mu^p_{\bar{k}_a} + \sum_{l \in dn(c) \setminus k_a} \phi^c_l) \\ \subseteq & \mathcal{S}[\phi^p_{\bar{k}_a} \cdot (\phi^c_c + \mu^p_{\bar{c}} + \sum_{l \in dn(c) \setminus k_a} \phi^c_l)] = \mu^p_{\bar{k}'_a}. \end{aligned} \tag{7.16}$$

Analogous reasoning leads to $X \subseteq \mu^p_b$ and we conclude that the matching condition as per Definition 11 is fulfilled.

Case $c = a$ or $c = b$. With respect to the former case, either k_a or k_b (but not both) do not exist. Without loss of generality, assume $c = a$. The induction for the summary ϕ^c_b still holds. Match $X \subseteq \phi^p_a \cdot \phi^c_b$ is found at a since $X \subseteq \phi^c_{\bar{k}_b} \cdot \phi^p_a$, and a is automatically notified of the match by Equation (7.10). To prove that b is notified too, one uses the slightly modified induction given by Equations (7.15) and (7.16). The base case, instead of Equation (7.15), is now:

$$\begin{aligned} X \subseteq & \phi^c_{\bar{k}_b} \cdot \phi^p_c \\ \subseteq & \phi^c_{\bar{k}_b} \cdot (\phi^p_c + \mu^c_{\bar{a}} + \sum_{l \in dn(a)\ k_b} \phi^p_l) \\ \subseteq & \mathcal{S}[\phi^c_{\bar{k}_b} \cdot (\phi^p_c + \mu^c_{\bar{a}} + \sum_{l \in dn(a)\ k_b} \phi^p_l)] = \mu^c_{\bar{k}_b}, \end{aligned} \tag{7.17}$$

and the rest of the inductive argument remains the same as before. Also here we conclude that the matching condition is fulfilled, and the proposition holds. □

Bloom Filtering for Strings

In this section we refine the matching algorithm for the summaries that explicitly have to do with character strings (that is, seq *CHAR*s). In the Section 7.2 the constraints were presented conceptually as subsets of the support set for a given coordinate. In general this description is not compact, as for a coordinate with the support set of k elements, there are 2^k possible constraints to encode. Some support sets have special structure that can be used to maintain a compact description even if the size of the support set is unlimited. This is precisely the case for strings, whose support set is infinitely large, and consists of the union of the sets of all character sequences with lengths 0, 1, 2 and so on.

Bloom filters [11] are a device to approximate set membership. A Bloom filter is a randomized bit array (it uses randomized hash functions) and has some possibility of yielding a *false positive*, that is it may posit that an element is in a set, where it is not [61]. The Bloom filter can also be designed to allow for some fixed fraction of false positives. The filter is most efficient if a positive implies lengthy processing, and the "positive" set is only a small fraction of the universal set. Even with false positives, the resulting set for which the lengthy operation is performed is small.

Bloom Filter Properties

Let there be given a length-m bit sequence B, called the Bloom filter.

$$B : \mathrm{seq}\{0, 1\}$$
$$\#B = m$$

Assume that there are n elements of some set S in total to represent by the bit array. Initially all the bits in the array are set to 0. A family of k hash functions is adopted:

$$[S]$$
$$h_1, \ldots, h_k : S \to 0..m-1$$

It is assumed that the hash functions are independent and produce a random distribution of the domain over the range $0..m-1$. The bloom filter is produced from the n elements of the set S by computing in turn the values $\{h_i(x) \mid i \in 1..k\}$ for each $x \in S$ and setting the bit $h_i(x)$ of the array B to 1. Testing whether some element q belongs to S amounts to computing $T = \{h_i(q) \mid i \in 1..k\}$ and checking that $0 \notin B(\!|\ T\ |\!)$.

The false positive probability is computed considering all the bits in B as independent. The probability that some bit j in the array is left at 0 after all the n elements of S are cached in B is equal to the probability that all the $n \cdot k$ bit settings set some bit other than j. As there are m bits in total, the probability of

one hash missing one particular bit is is $m - 1/m = 1 - 1/m$. As there are $n \cdot k$ bit settings, the probability that bit j remains unset is equal to the probability that all the bit settings miss. That is:

$$\left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}. \tag{7.18}$$

The probability of a false positive is equal to the probability that for the element q all the k hashes evaluate to set bits of B. Thus the false positive probability is approximated by:

$$\left[1 - \left(1 - \frac{1}{m}\right)^{kn}\right]^k \approx \left(1 - e^{-kn/m}\right)^k = f. \tag{7.19}$$

Taking the right-hand side of Equation (7.19), it can be noted that the false positive probability for the Bloom filter as given here depends on the three parameters: k, m, and n. One may consider the choice of k for fixed m and n so that the false positive probability f is minimized. In that case, f is minimized easily considering that it shares a common minimum over k with $\ln f$:

$$\frac{\mathrm{d} \ln f}{\mathrm{d}k} = \ln e^{-kn/m} + \frac{kn}{m} \frac{e^{-kn/m}}{1 - e^{-kn/m}} = 0. \tag{7.20}$$

The Equation (7.20) is satisfied for $k = m/n \cdot \ln 2$ which can be readily checked[8], and this also constitutes a global minimum. In practice however, $\lceil k \rceil$ or $\lfloor k \rfloor$ are applicable, as k is in fact constrained to integer values. Further, minimizing f in terms of $p = e^{-kn/m}$ when k is minimum reveals that $p = 1/2$ minimizes f [61]. Thus, the Bloom filter performs best when the probability that each bit is set is at $1/2$. As the bits are independent, the sequence B then looks like uniform random (binary) noise.

Counting Bloom Filters

A drawback of the original Bloom filter is that the element removal from the filter is not supported. A moment's thought reveals that simply setting indices of $h_i(q)$ for an element q to 0 does not work as intended. Resetting these bits effectively excludes many more elements from the Bloom filters, and precisely those that have *any* bits that coincide with any $h_i(q)$. This is a large set because, opposed to the original formulation of the filtering where *all* bits must be set for an element to be a member, in this case *any* element may be reset for an element to not be considered a member. The probability of this happening is now about $1 - m^{-n}$, which quickly gets close to 1, for increasing n.

[8] This claim can be derived by substituting $t = e^{-kn/m}$ and massaging Equation (7.20) until the equation $(1 - t)\ln(1 - t) = t \ln t$ is reached. From there can be concluded that $1 - t = t$, hence $t = 1/2$. Returning to the substitution yields the result.

To support element removal from the filter, a *counting* modification is introduced to the original filter definition. It replaces each bit of the original filter with a counter. Adding an element q to a filter increments each counter in the index $\{h_i(q) \mid 1 \leq i \leq k\}$. Removing an element q from a filter decrements each counter in the respective index set. Additionally, for space considerations, and for practical purposes, all counters have a limited range, from 0 to some maximum w. For this reason, the increment and decrement operations are saturating. Thus incrementing w yields w instead of $w + 1$. Likewise decrementing 0 yields a 0 instead of -1. The saturation does not introduce errors in the filter operation. It rather only prevents subsequent removal operations to restore the filter set exactly.

Applying Bloom Filters in the Matching Algorithm

To apply the (counting) Bloom filter for the matching algorithm, one needs to overload the union and intersection operators for the Bloom filter type. We introduce a derivative of a *Constraint*, which is a counting Bloom filter. The newly defined *Constraint* and the related operations given below can be readily plugged into the previously defined matching algorithms.

$$
\begin{array}{|l}
\textit{BloomConstraint} \\
\hline
b : \operatorname{seq} 0..m-1 \\
h_1, \ldots, h_k : \operatorname{seq} \mathit{CHAR} \rightarrow 0..m-1 \\
\hline
\#b = k \\
\hline
\end{array}
$$

Testing for membership is achieved by testing for the absence of the element 0 in the relational image of the element hash functions.

$$
\begin{array}{|l}
[X] \\
\hline
\_ \in \_ : X \leftrightarrow \textit{BloomConstraint} \\
\hline
X \neq \operatorname{seq} \mathit{CHAR} \bullet \\
(\_ \in \_) = \varnothing \\
X = \operatorname{seq} \mathit{CHAR} \bullet \\
\quad \forall\, x : X;\ y : \textit{BloomConstraint} \bullet \\
\qquad 0 \notin y.b(\!| \{y.h_i(x) \mid 1 \leq i \leq k\} |\!) \\
\qquad\quad \Leftrightarrow (x, y) \in (\_ \in \_) \\
\hline
\end{array}
$$

Further, computing union and intersection amounts to maximizing, or minimizing the value of the respective counter. The intersections with other types are all empty, while unions are undefined.

$$\_ + \_ : BloomConstraint^2 \rightarrow BloomConstraint$$
$$\_ \cdot \_ : BloomConstraint^2 \rightarrow BloomConstraint$$

$$\forall\, x, y, z : BloomConstraint \bullet$$
$$z.b = \langle max(x.b(1), y.b(1)), \ldots, max(x.b(k), y.b(k)) \rangle \Leftrightarrow (x, y) \mapsto z \in (\_ + \_)$$
$$z.b = \langle min(x.b(1), y.b(1)), \ldots, min(x.b(k), y.b(k)) \rangle \Leftrightarrow (x, y) \mapsto z \in (\_ \cdot \_)$$

Matching Implementation

In this section we present the implementation of the matching algorithm described in Section 7.3. The implementation is presented in the form of an appropriate CPN performing steps described in the algorithm description. The CPN reveals the flow of control for the implementation. The synchronization between the places and transitions functions according to the execution rules for the CPN. The account of these rules was given in detail in Chapter 2.

The CPN implementing the matching is shown in the Figure 7.3. It is a straightforward rewrite of the matching algorithm into the CPN form. The Figure shows the places through which the specification and match tokens go within a single agent. Using the *folding* technique the same CPN is used to specify the states of all the nodes in the network. Places U^*, D^* and L^* denote the set of *uplinks*, *downlinks* and *total links*, respectively. The contents of these places depends on the environment. The *uplink* places contain the uplink for every node. The *downlink* places contain the downlinks for every node. We assume that the external mechanism exists that maintains the current view of U^* and D^*. This function is performed by the CBT package (see Chapter 6). Additionally, the assertion $L^* = D^* \cup U^*$ holds, as uplinks and downlinks are also links[9].

At the top of the CPN of Figure 7.3, one notes three "phantom" transitions: ($recv_m$), ($recv_s$) and (*local*). These are the interfaces to the mechanisms that obtain the remote matches, remote specifications and the local specifications, respectively. The remote matches correspond to match messages (see Equation 7.12) obtained from the uplinks k for the node n. The remote specifications correspond to the specifications obtained from downlinks (see Equation 7.11). Finally, the local specifications are those published at the local node n, signifying the messages the node n is interested in. The same algorithm operates on both the *producer* and the *consumer* summaries. This is denoted by an upper index x (where $x \in \{p, c\}$) of the respective specifications and matches. Hence the given algorithm can be understood as *two* superimposed CPNs, one for $x = p$, and the other for $x = c$. These algorithms are not completely independent however, as in

[9]That is, the implementing classes for *downlinks* and *uplinks* are derived from a common base class *Link*.

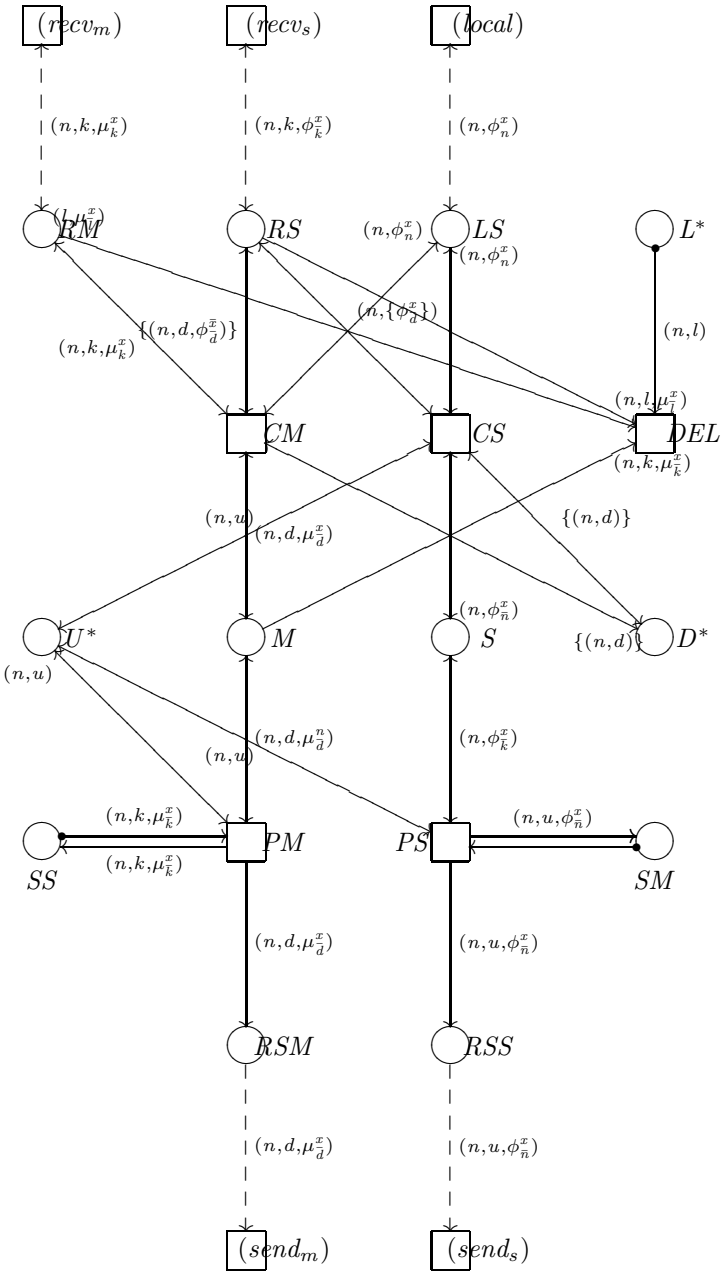


Figure 7.3: The CPN showing the matching algorithm.

the Equation 7.12, the complementing summaries, x and $\bar{x}$ appear. Still, the two CPNs are similar enough so as to be folded to a single net, given in Figure 7.3.

The remote and local summaries are collected in the places: Remote Match (RM), Remote Summary (RS), and Local Summary (LS). For each node n respectively, these places hold:

1. The matches received from another node k;
2. The summaries received from another node k; and
3. The local summary generated at the node n.

The transitions Compute Match (CM) and Compute Summary (CS) respectively compute the Equation 7.12 and Equation 7.11. The transition Delete (DEL) removes the stale matches and links when due to the environment influence a link (i.e. a token on Link (L^*)) is lost.

The transitions CM and CS produce the computed Match (M) and Summary (S) tokens which regularly get updated as the specifications on RM, RS and LS change. Package Match (PM) and Package Summary (PS) package and ship out the readies summaries to Ready Match (RSM) and Ready Summary (RSS). These last tokes are picked up by ($send_m$) and ($send_s$) and are being transferred to the respective destinations. The destinations are shown in the CPN as d and u respectively. Sent Summary (SS) and Sent Match (SM) maintain a tally of the summaries already sent and will prevent the sending of new summaries for as long as no changes to the summary occurred since it has been last sent.

The details of the token transfer between ($send_x$) and the corresponding ($recv_x$) are omitted in the presentation of the algorithm. This is because the dialog mechanics has already been described before (e.g. Chapter 6). We therefore need only to mention a simplified model of the token transfer. The sent triples at ($send_x$) for $x \in \{m, s\}$ always have the form $t = (s, d, c)$, where s is the source node identifier, d is the destination node identifier and c represents the content of the transferred summary. At destination (i.e. ($recv_x$)) the same triples are received as $t' = (d, s, c)$. As can be seen, $\mathit{first}(t)$ and $\mathit{first}(\mathit{rest}(t))$. are transposed in the received copy t' of t.

7.4 The Workflow Mechanics

In the previous sections, we described the data model used for the workflow execution (Section 7.2), and then we described the mechanism used for matching (Section 7.3). In this section we bring these elements together, describing how the workflows are maintained by using them and how multiple function instances coexist when they are physically located at different nodes.

The principal task of the workflow mechanics is to explain the way the dataspace and subspace selection can be used for flow control in a distributed workflow execution system. In the Chapter 4 it was shown how inter-related tasks can be

cast into a workflow form by defining *guard* conditions which determine the necessary and sufficient conditions for a task to be activated. It was also demonstrated how such a specification is cast in terms of a CPN. From there we concluded that a coordinated task set can be expressed in terms of an appropriately annotated CPN. On the other hand, in the previous Sections it is shown how nodes can use the constraints to select only the parts of the *Dataspace* for which they explicitly express an interest in. The underlying thread in this Section is the provision of a mechanism that can implement a CPN in terms of the operations available on the *Dataspace* elements.

Versioning

A part of this task is the assignment of each CPN transition to a node that executes the corresponding program code, and using the suitable producer-consumer summaries to filter out from the *Dataspace* the appearance of only the appropriate tokens that activate the transitions. Hence, for any CPN an appropriate set of producer-consumer summaries need to be found, selecting only the elements appropriate for a transition.

The network and node volatility is the main issue. Not only can the node outage prevent a workflow component to be executed (if not handled properly), it can also cause the system to lose track of the operations that were unfolding. The nodes in the system must be able to transport the tokens from one to the other, and also make a tally which tokens have been transported already. Hence for all tokens, additional bookkeeping is needed to achieve this. Unfortunately, this is not achievable through flagging the delivered objects locally. This is because such a marking does not confirm that an object has in fact been delivered and processed. Also, as multiple nodes may perform equivalent functions, it is possible that due to *aliasing*, multiple equivalent objects are produced by different nodes. There is no need to deliver multiple such objects (rather only one or two, for instance). Hence flagging the objects as delivered would produce a system state where there exist equivalent objects, of which one is flagged and the rest are not, although *all* object instances should be considered delivered.

Flagging is *not* enough to ascertain that some object is delivered. This is because the "deliveredness" of an object can be determined only if the joint states of both the producers and consumers are considered. Flagging an object as delivered amounts to estimating the state of the consumer as being "object has been received", and the estimate can of course be wrong. On the other hand, the summaries used to determine the compatible interests can remain unchanged when nodes join or leave the CBT. This is because the summaries at a node are always obtained by computing a superset of the summaries forwarded by the neighbours. In both cases, counters are required to denote the situations in which a change to the object has occurred, without actual change on the object itself (i.e. a new, identical, object instance has become available), or that a summary has changed its composition, without changing the *Dataspace* subset it refers to.

The said issues are handled by introducing a *version* annotation for both the objects and the summaries themselves. The object and the summary annotations are somewhat different one from the other. The difference stems from the fact that a summary is unique per CBT, whereas an object can appear accross the nodes in multiple equivalent instances.

Version

$tag : \mathbb{R}$

Object Versioning

For the object versioning, the version tag must fulfill two requirements:

1. *Version distinction.* Different versions of an object must be such that, clearly, the object history can be derived from the versioning number.

   That is, by comparing the version tags of two equivalent objects which are not the same version, it is possible to determine which one of them is an "earlier" and which one is a "later" version.

2. *Object distinction.* Different object instances with the same versions need to be distinct from one another, but still be selectable based on the *tag* identifier of the associated *Version* instance.

   That is, by comparing the version tags of two equivalent objects with the *same* version number, it is possible to establish an ordering based on the version tag alone.

Adopting a real-numbered version tag handles these two issues. The total ordering with respect to the relation $\_ \leq \_$ is established on the set $\mathbb{R}$, hence the version numbers can be readily compared. As $\_ \leq \_$ is also defined on the positive integers (the set $\mathbb{N}$), we observe that for the fulfillment of the requirement 1, using $\mathbb{N}$ for this purpose would be just enough.

However, in that case, all the instances of an object with a particular version tag would become indistinguishable. This would mean that a request for a versioned object would yield potentially many hits, whereas only a single object would suffice. For this reason, the following rule is adopted:

Proposition 8 (Object versioning) *Let there be given an object o annotated with a version $v \in Version$.*

1. Version distinction *Each object has a Version annotation containing a tag: $tag \in \mathbb{R}$. The version of an object $o\langle v = \langle tag \Rrightarrow x \rangle\rangle$ is therefore given as $\lfloor v.tag \rfloor$.*

2. Object distinction *Each object instance version is increased by the fraction*[10] *$\{v.tag\}$ an unique pseudo-random number from the interval* 0..1.

[10] *We use the notation from [37] which introduces the "decimal fraction" operator $\{x\}$ by defining $\{x\} = x - \lfloor x \rfloor$.*

Example 22 (Object versioning) *Let there be given some object $o \in Dataspace$, with version number* 2*, and let α and β be the nodes at which the object instances reside.*

The instances are referred to as $o \diamond \alpha$ and $o \diamond \beta$. When the object o is stored at two nodes α and β, it is assigned a version number 2 *increased by a pseudo-random value in the interval* 0..1*. Hence the object instances would be referred to as:*

$$o \diamond \alpha\langle v = \langle tag \Rrightarrow 2.1362\rangle\rangle$$

and:

$$o \diamond \beta\langle v = \langle tag \Rrightarrow 2.7213\rangle\rangle$$

respectively.

The randomized versioning ensures that the ordering between object instances is preserved, and that the objects remain distinguishable within the same version number. It is therefore possible for the consumers to specify a base version of some wanted object o, and to also tune which version range they are interested in. A consumer interested in a version t of o can start by requesting o with a version tag that satisfies the predicate $(\_ \leq 2.1)$.

If, after some predefined time interval, the consumer does not receive such an object, it can modify its summary to look for version tags that are the members of the set defined by the unary predicate $\_ \leq 2.2$, and so on, until a suitable object is found. The consumer can hereby gauge the stream of the delivered objects so that the network is never overwhelmed with the delivered object copies. Also, eventually the object with version tag 2 will be delivered to it, if it exists.

Proposition 9 (Copying rule) *Whenever an object $o\langle v\rangle$, with $v \in Version$. is copied as a result of any operation (e.g. forwarding), the version annotation $v.tag'$ of the copy is set to:*

$$v.tag' = \lfloor v.tag + 1 \rfloor + rnd,$$

where rnd is a pseudo-random number from the interval 0..1*.*

In this way, the object instances with an integer version tag $\lfloor x \rfloor$ get assigned a version number x which is between $\lfloor x \rfloor$ and $\lfloor x \rfloor + 1$. It is straightforward to determine the object instance version tag distribution in this interval.

Typically, a node requesting an object will try to obtain the most recent version thereof, and as few redundant copies as possible. Ideally, only one object instance will be received. A node can therefore tune its selection policy so that with high probability only a single object instance is delivered, and as recent a version as possible. To achieve this, it is an advisable strategy for a node to tune its constraint policy so that on average, a single object is delivered to it. The node selects the value $\lfloor x \rfloor$ as the version number. The next step is to choose $\{x\}$ that yields one object as a result.

Proposition 10 (Versioning rule) *Let $\lfloor x \rfloor$ be the version number of an object a node has an interest in. Assume that the number of pending objects between $\lfloor x \rfloor$ and $\lfloor x \rfloor + 1$ is known to be n. Then the choice of $\{x\}$ such that the version constraint $x = \lfloor x \rfloor + \{x\}$ yields a single object with the highest probability:*

$$\{x\} = 1 - e^{-1/n}. \tag{7.21}$$

Proof. Assume that n object instances are posted in the interval between $\lfloor x \rfloor$ and $\lfloor x \rfloor + 1$. For brevity introduce shorthand $q == \{x\}$. Let $X(q)$ be a random variable denoting the number of the n objects with the version tag falling between $\lfloor x \rfloor$ and $\lfloor x \rfloor + q$.

Then the probability that only a single object falls into this interval is:

$$\Pr\left(X(q) = 1\right) = nq(1-q)^{n-1}. \tag{7.22}$$

The behaviour of $\Pr\left(X(q) = 1\right)$ on the set $[0, 1] \times [1, 100]$ is shown in Figure 7.4. This probability reaches the extremal value at the same point as $\log \Pr\left(X(q) = 1\right)$.

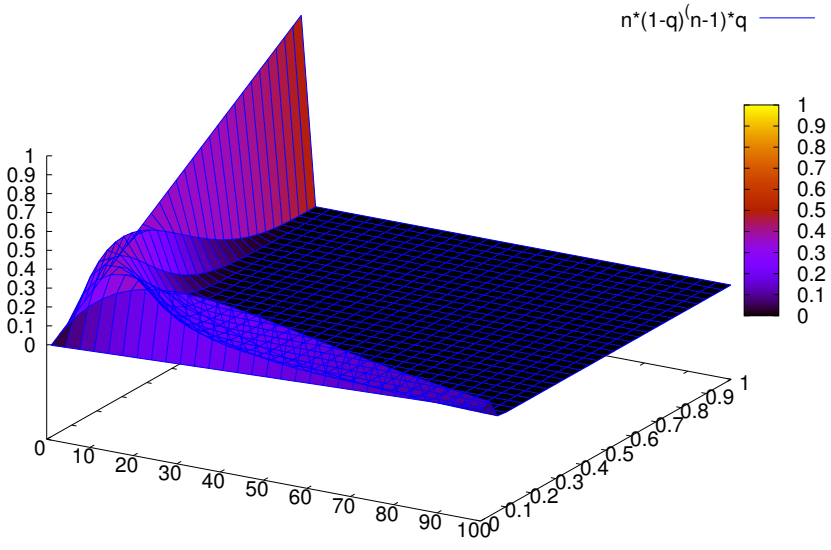


Figure 7.4: Behaviour of the probability $\Pr\left(X(q) = 1\right)$.

$$\frac{\mathrm{d}}{\mathrm{d}n} \log \Pr\left(X(q) = 1\right) = \frac{1}{n} + \log(1-q) = 0, \tag{7.23}$$

from where it is concluded that:

$$q = 1 - \exp(-1/n)$$

maximizes the probability that $X(q) = 1$, which then amounts to:

$$\Pr\left(X(q) = 1\right)\big|_{q=1-e^{-1/n}} = ne^{-1/n}(1 - e^{-1/n}).$$

□

Hence, for selecting with highest probability an interval that contains a single object, nodes should estimate the number of candidate objects n, and then set the fraction $\{x\}$ to the value given in the Equation 7.21.

As n is typically unknown, we must settle for an estimate for n. The estimate can be derived as follows, from the past observations of the same version range for one particular object instance.

Proposition 11 (Estimating instances) *Assume that for r various values of $\{x\}$ being q_1 up to q_r, the following samples are obtained: n_1, n_2 up to n_r. Then the estimate for the total number of objects within the range* 0..1 *is:*

$$n = \frac{1}{r} \sum_{1 \le i \le r} \frac{n_i}{q_i}. \tag{7.24}$$

Proof. The probability for the realization of the event $(n_1, \ldots, n_r)$, where the samplings of n_i are independent for all i is:

$$\Pr\left(n_1, \ldots, n_r\right) = \prod_{i=1}^{r} \binom{n}{n_i} q_i^{n_i} (1 - q_i)^{n - n_i}. \tag{7.25}$$

The likelihood function is obtained as: $L(n_1, \ldots, n_r) = \log \Pr\left(n_1, \ldots, n_r\right)$. Differentiating L with respect to n:

$$\frac{\mathrm{d}L(n_1, \ldots, n_r)}{\mathrm{d}n} = \frac{\mathrm{d}}{\mathrm{d}n} \sum_{1 \le i \le r} \log \binom{n}{n_i} + n_i \log q_i + (n - n_i) \log(1 - q_i) = 0. \tag{7.26}$$

The log-binomial $\log \binom{n}{n_i}$ is expanded, taking into account that $\binom{a}{b} = \frac{a!}{b!(a-b)!}$ for $a \ge 0$, $b \ge 0$ and $a - b \ge 0$, to yield:

$$\begin{aligned} L(n_1, \ldots, n_r) = \sum_{1 \le i \le r} \left(\sum_{1 \le j \le n} \log j - \sum_{1 \le j \le n_i} \log j - \sum_{1 \le j \le n - n_i} \log j \right) \\ + n_i \log q_i + (n - n_i) \log(1 - q_i). \end{aligned} \tag{7.27}$$

The sum-logs on the right-hand side of Equation (7.27) are approximated by the Stirling formula:

$$\sum_{1 \le k \le x} \log k \approx \int_1^x \log k \mathrm{d}k = x \log x - x, \tag{7.28}$$

to yield the approximation:

$$L(n_1, \ldots, n_r) \approx \sum_{1 \leq i \leq r} n \log n - n_i \log n_i - (n - n_i) \log(n - n_i) + n_i \log q_i + (n - n_i) \log(1 - q_i). \tag{7.29}$$

Differentiating the approximate expression for L from Equation (7.29), one obtains the condition:

$$\frac{\mathrm{d}L(n_1, \ldots, n_r)}{\mathrm{d}n} = r \log n - \left[\sum_{1 \leq i \leq r} \log(1 - q_i) + \log(n - n_i) \right] = 0 \tag{7.30}$$

that reduces to:

$$\prod_{1 \leq i \leq r} (1 - q_i) = \prod_{1 \leq i \leq r} \left(1 - \frac{n_i}{n}\right). \tag{7.31}$$

The Equation (7.31) is satisfied for:

$$n = \frac{n_i}{q_i}, \text{for all } 1 \leq i \leq r. \tag{7.32}$$

Adding all the expressions $n = n_i/q_i$ from Equation 7.32 up yields:

$$rn = \sum_{1 \leq i \leq r} \frac{n_i}{q_i}, \tag{7.33}$$

leading to Equation (7.24). □

From Proposition 11 we see how a node can estimate the number of object instances, given past observations. Thus obtained estimate for n can then be substituted into the Equation (7.24) to yield the size of the selection $\{x\}$.

This versioning alone is not enough for the correct selection of the desired objects from the neighbouring nodes. As with objects, the summary versions play an important role, and we therefore treat their versioning in the following Section.

Summary Versioning

As with the objects, the summaries delivered between nodes must be versioned. The versions on summaries are required to propagate the changes to the producer and the consumer sets as well as the changes of the summaries themselves.

As an illustration, consider the society given in the Figure 7.5 showing the interaction between three nodes named α, β and γ. The nodes are represented by circles, and the summaries reported to uplinks are represented by the annotated summaries ϕ. Figure 7.5 shows a simple scenario in which a new node joins an already existing CBT structure for which a summary ϕ has already propagated

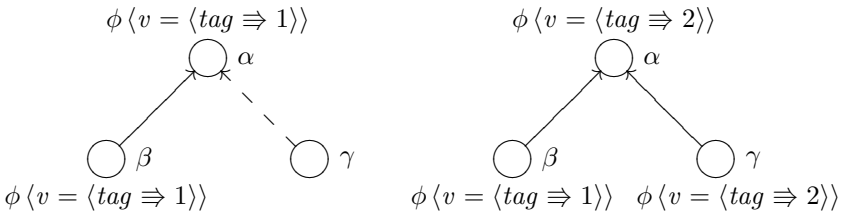


Figure 7.5: The need for summary versioning.

through (left). Incidentally, the node γ contributes a new summary which denotes the same *Dataspace* subset as those of the nodes α and β.

Since the summary of γ is also equal to ϕ, without additional provisions, α would not need to modify its aggregated summary. But, although not changing the summary is economic in terms of the bandwidth saved for not transferring an unchanged summary, it also entails that γ will not be delivered any objects present on α which have already been delivered for the summary version 1 of ϕ during the time γ was not included as a consumer.

To do this, however, the node γ must specify a version tag that is higher than that present in the uplink. It is not always necessary that γ requests all the objects for its particular summary. This depends on whether γ is introduced to take up the upcoming workload (from the time it joins onwards), or it is introduced to take over the objects posted before it joined the network.

When computing the summaries (according to Proposition 7) in the CBT structure, the node[11] b annotates the resulting summary with a version tag that is the maximum of all the version tags of the constituting summaries.

Proposition 12 (Versioning summaries) *Let b be some node in the CBT. Let $dn(b)$ be the set of its downlinks. Let b compute the summary to be forwarded to $u = up(b)$ according to the Equation* (7.11)*. Additionally, let each summary ϕ^x_k for all $k \in dn(b)$, as well as ϕ^x_b be annotated with the corresponding version tag. Let these annotations be $\phi^x_b.v$ and $\phi^x_k.v$ for all $k \in dn(b)$, respectively.*

Then the annotation for ϕ^x_b is obtained as:

$$\phi^x_b.v = \left\langle tag \Rrightarrow max\left[\phi^x_b.v.tag, max_{k \in dn(b)}\, \phi^x_k.v.tag\right]\right\rangle \tag{7.34}$$

From the Equation (7.34) we see that each node b forwards a maximum version tag of the subordinate summaries along with the summarized tag. This tagging ensures that any change in the number of downlinks along a thread[12] gets eventually propagated up a thread, as long as the version tag associated to the change is the maximum along a thread.

[11] Refer to Proposition 7 on page 172 for details about the notation.
[12] I.e. those connected by the *uplink* and *downlink* relations.

A similar rule is valid for the match computation. Here the version tag of the computed match is again the maximum of the version tags of all the summaries used to compute it.

Proposition 13 (Versioning matches) *Let b be some node in the CBT. Let $dn(b)$ be the set of its downlinks. Let b compute the summary to be forwarded to $k \in dn(b)$ according to the Equation (7.12). Additionally, let each summary $\phi^{\bar{x}}_{\bar{k}}$ for all $k \in dn(b)$, as well as the summaries ϕ^{x}_{b} and $\phi^{x}_{\bar{k}}$ be annotated with the corresponding version tags. Let these annotations be $\phi^{x}_{b}.v$ and $\phi^{x}_{\bar{k}}.v$ for all $k \in dn(b)$, respectively.*

Then the annotation for $\mu^{x}_{\bar{k}}$ is obtained as:

$$\mu^{x}_{\bar{k}}.v = \langle tag \Rrightarrow max[\phi^{x}_{\bar{k}}.v.tag, \phi^{\bar{x}}_{b}.v.tag, \mu^{x}_{\bar{b}}.v.tag, max_{l \in dn(b) \setminus k}\, \phi^{x}_{\bar{l}}.v.tag] \rangle \tag{7.35}$$

The node b of the CBT can therefore specify a version tag through annotating its client summary, and can observe the effect of putting an annotation into the summary by testing the version tag on the respective match. Depending on the intended role of the node, the node may want to impose the maximum version tag for its thread. This happens if the node should collect the objects that had been exported by other nodes in the CBT prior to its activation in the CBT. If the node is included only to continue the work from a certain point onwards, it need only set its version tag to be equal to that of the current match.

Hence, the strategy for a node that establishes a new connection with an uplink in its CBT is to select a version tag for the summary higher than the previous one for the respective match. And the strategy for a node that reinstantes an existing connection is to set a version tag *higher* than that of the current match.

With the object and the summary versioning in place, the bookkeeping of the object delivery is simple. Per object, a *last delivered* version tag is kept. It denotes the last match version for which an object copy was delivered.

$$LastDelivered == Version$$

The delivery is directed by the following simple rule.

Proposition 14 (Delivery rule) *Consider an object $o\langle l = \langle tag \Rrightarrow l^* \rangle \rangle$ and a match $\mu\langle v = \langle tag \Rrightarrow v^* \rangle \rangle$. Then the LastDelivered tag of o is updated, and the o forwarded to the owner of μ by:*

$\Delta LastDelivered$

$v^?? : \mathbb{R}$

$tag < v^?? \bullet$
$\quad deliver\ o\ for\ \mu.$
$\quad tag' = v^??$

As the matches consist of disjoint cubes, there will be at most a single cube C within the match μ for which $o \in C$. Hence object is delivered only once for a single node, per each version change that ensures that $l^* < v^*$.

Waiting on Multiple Tokens

In Chapter 2 the triggering mechanism for transitions was described. The triggering mechanism takes care that each CPN transition (implemented by a node in the CBT) gets activated if and only if the preconditions for this are fulfilled. The triggering mechanism must also take care that the tokens fulfilling the precondition for some transition are processed only once, provided that no version changes on these objects have taken place.

In line with the requirement to handle the *node volatility*[13], multiple instantiations of the same transition must be supported. By *multiple instantiations* of a single transition, we mean multiple nodes which implement functionally identical transitions. Supporting multiple instantiations translates into the bookkeeping required to direct all the available tokens only to a specified set of receivers. To understand this properly, we take a look at the components that any workflow consists of, shown in the Figure 7.6.

Depending on the number of the input and output places we distinguish the following CPN basic building blocks:

1. *Contract* is the building block consisting of a transition t, where $\#(\circ t) > 1$, and $\#(t\circ) \leq 1$ (when $\#(t\circ) = 0$, then t is also a *sink*). Upon each activation, the contract transition reduces the total number of active tokens (as the number of input places is smaller than the number of output places). According to the activation rules from Chapter 2, the token types b and c must depend on only one of the other tokens (say a). Hence: $b = b(a)$, and $c = c(a)$.

2. *Expand* is the building block consisting of a transition t, where $\#(\circ t) \leq 1$, and $\#(t\circ) > 1$ (when $\#(\circ t) = 0$, then t is also a *source*). Upon each activation, expand increases the number of active tokens.

3. *Reduce* is the building block that consists of a transition t, which produces and consumes tokens from the same place. Such a transition can occur typically for cumulative operations, arising from say summation.

4. *Map* is the building block for which $\#(\circ t) = \#(t\circ) = 1$. This block consumes a single token from the only place in $\circ t$, and delivers it to the only place in $t\circ$.

Note that the blocks for which simultaneously $\#(\circ t) > 1$, and $\#(t\circ) > 1$ can be described by a serial concatenation of an *expand* block with one *contract* block,

[13] See page 10 on node volatility.

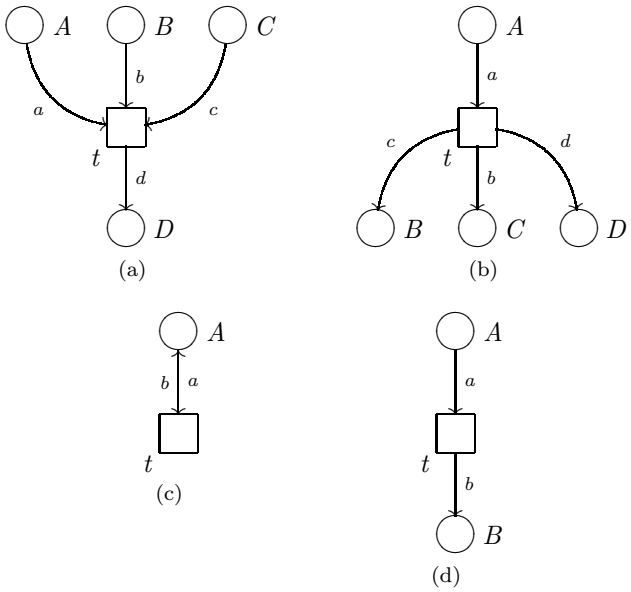


Figure 7.6: The basic CPN building blocks. (a) Contract. (b) Expand. (c) Reduce. (d) Map.

by making the unique post-place of the latter the unique pre-place of the former (see Figure 7.7). The access modes[14] are syntactic sugar so there is also no need to show them here explicitly.

When the CPN is implemented, each transition is allocated to some node. To prevent problems relating to volatility, as well as to enhance throughput, the system designer would want to allocate multiple instances of the same transition in different nodes. This gives rise to the need to synchronize the instantiations, so that the activation semantics of the CPN can be honored. For this, some distributed queuing method for each transition needs to be applied.

Various strategies exist for this [44]. The first option is centralized queue regulation through an unique process, the *queue manager*. While simple, this queuing solution has two important drawbacks:

1. With the increase in the number of queues to manage, the manager ends up with a high workload. It ultimately becomes a bottleneck.

2. Under network and node volatility, the queue manager also becomes a single point of failure, whose absence prevents the CPN execution.

[14]The access modes were shown in Chapter 2, on page 31.

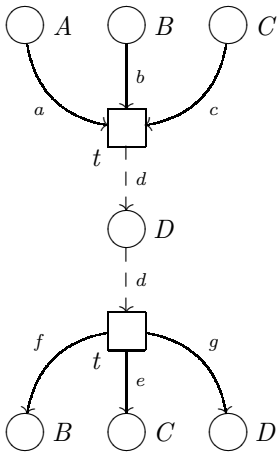


Figure 7.7: Attaching an *expand* to a *contract* building block.

The second option is having each producer multicast the state of the tokens it contains to all the consumers. Again, the simplicity of the scheme is outweighed by the two principal drawbacks:

1. This approach entails a lot of messages for synchronization.
2. Care must be taken that each token offered by a producer is consumed only once in a given context.

A separate issue arises with the contract basic building blocks. A contract building block (call it k) waits on multiple tokens originating from disparate nodes. At the same time, the block k can have multiple instantiations residing on different machines. Upon the delivery of the tokens in the set $\circ k$ it must be ensured that all tokens relevant for k are not only delivered to an instance of k, but also that *all* tokens of the places from $\circ k$ are delivered to the *same* instance of k.

The ideal approach for the synchronization of multiple producer-consumer instances forwards the tokens (i.e. the products) so that all the consumer instances are busy processing the tokens and that some goal function is optimized. Typical goal functions are *balancing* the load on the consumers, *maximizing* the cumulative token throughput, or minimizing the time for processing a single token.

We do not go into the ways the execution is optimized. Here we only treat the mechanism with correct token delivery as the only functional requirement. On the non-functional side though, the following requirements are important:

1. The avoidance of centralized setups (e.g. the queue manager approach given above). This requirement stems from the network volatility assumption. In a volatile network, it is not possible to employ an effective queue manager,

since it is as susceptible to volatility as any other process. It therefore remains a bottleneck for the entire system, regardless of the existence of the multiple instances of the consumers.

2. Proactive handling of the node volatility. This requirement specifies an approach to handling node volatility before it occurs.

   While it is considered an overkill in computer systems with assumed infrequent fault, it is quite a natural assumption in the communication systems, for instance. As we identified node volatility to be ever-present in DWEAM, proactive handling of node volatility is justified.

Mutual Adjustment Strategy

From the above requirements, the coordination strategy of *mutual adjustment* [82] between the nodes implementing transition instances is imminent. The mutual adjustment is a coordination strategy used in distributed systems, whereby the system components cooperate in managing their own workload, without an external supervisor entity. The mutual adjustment strategy employed here is based on the *problem space partitioning* concept. This strategy is used given that, due to the task decomposition into a workflow and the representation by CPN transitions, the transition instances apply identical procedures to the tokens in the input place. As opposed to the queue manager approach, where the queue manager explicitly decides which worker takes a work item, we adopt an implicit approach, in which no queue manager exists, and the task assignment is randomized.

We introduce a simple mutual adjustment schema based on the assignment of unique identifiers to points of the *Dataspace*, and partitioning the identifier space among all the interested transition instances. Trading off top performance for coordinator-less operation, we obtain a strategy that guarantees eventual progress of the involved tokens, if the processing is feasible (i.e. at least one appropriate transition and the network connection exist). For this purpose an additional coordinate is needed in the *Dataspace*. We call this coordinate the *DSUID*. It is used as a mutual adjustment key.

[*DSUID*]

The *DSUID* is an unique identifier type, from which elements can be drawn randomly. Further assume for simplicity that each transition of the CPN to be executed has been allocated a distinct numeric *DSUID* too (presumably by the system designer).

A straightforward way to implement *DSUID* is to represent its elements by a sequence of l digits taken in some base b. In practice, $b = 2$ is usually used (corresponding to using the identifiers encoded in binary), with l an implementation parameter, chosen so that b^l is large enough to contain all the objects in the

Dataspace[15]. An element $e \in DSUID$ is sampled from *DSUID* by generating uniformly at random each of the l sequence elements. As a result, set of the tokens from *Dataspace* will have *DSUID* coordinates that also uniformly at random fill the *DSUID* space.

To partition the *DSUID* set and allocate each partition element to some instantiation i of a transition t, distinction must be made between instances $t \diamond i$ for each i. For this, each node hosting an instance $t \diamond i$ should choose own identifier, from which the transition identifier can be recovered, and the identity of $t \diamond i$ can be maintained. Both goals can be achieved with a single identifier, as long as the upper bound on the number of functions and the number of implementing instances is known.

One can employ a simple idea coming from the coding for telecommunications to achieve this goal. The transition instance identifier is built from two components. The first is fixed, and corresponds to the transition identifier. One can look at this as a "codeword". To this codeword, an unique, randomly generated identifier is superimposed, by bitwise addition, considering both identifiers as binary numbers. The randomly generated identifier corresponds to random "noise" which is superimposed to a codeword when it is transferred through a communication channel. Well known coding schemes exist (Reed-Solomon, BCH, Turbo, Low Density Parity Check Codes, to name a few) by which the original codeword can be recovered efficiently from the "noisy" codeword.

In a similar manner, here the effect of noise is simulated, with the goal to obtain unique identifiers, distinct with high probability, but which cluster around the transition codeword, in the sense of some distance measure. A simple measure such as Hamming distance[16] suffices. From the unique identifiers, the transition codeword can be recovered by standard decoding methods. Contrary to the communication case where the noise is a feature of the communication medium, the added noise in our case is under direct control. Care must be taken to make the noise of low enough power so to not disturb the decoding by introducing spurious errors. The random identifiers can be generated completely independently by the implementing nodes.

Once the instance identifiers are assigned, a simple mutual adjustment strategy can be employed: a produced object is delivered to all different transitions that consume it, and within the transition, to the transition instances with *DSUID*s closest to the *DSUID* of the object itself. The system designer may decide to instruct the system to deliver the object to $c > 1$ closest transition instances, thus ensuring that the outage of any one single instance does not affect the subsequent availability of the computation outcome. We arrive at a delivery proximity rule.

[15]This is easily achieved in practice, as for instance if $b = 2$ and $l = 400$, then $b^l \approx 2.5 \cdot 10^{120}$, generously topping the number of hydrogen atoms in our galaxy.

[16]Hamming distance between two binary numbers x and y is equal to the number of bit positions in x and y that have different value.

Proposition 15 (Delivery Proximity Rule) *Objects representing the tokens in the CPN are annotated with a randomly assigned DSUID, as described. Instances that implement transitions are assigned a randomly generated DSUID as described. The token objects, when produced, are delivered to the c of the transaction instances which have the c closest DSUIDs to that of the token object.*

Multiple Input Tokens

For transitions that expect tokens from at most a single place (such as the *map* and *expand* from Figure 7.6b,d), the mutual adjustment strategy as described by the previous section can be considered enough to ensure that a transition is eventually delivered the tokens needed for firing.

With multiple tokens, as is the case of the *reduce* (Figure 7.6a), the mutual adjustment entails a complication: the input tokens, which can originate at unrelated nodes, must be delivered to the same transition instance. It means that the unrelated tokens must bear the same *DSUID*, so that the view on the closest transition instance is the same for all the tokens, and that token *DSUID*s must be delivered to all of them. A possibility for brokering the *DSUID*s in such a way is to assign them upstream (in terms of the defining CPN) in a transition common to all the tokens. The brokering example is given in Figure 7.8.

Such an arrangement implies that, for determining the transition instance at the minimum-distance from a given object, the instance *DSUID*s must be known. The *DSUID* information can be readily attached to the producer and consumer summaries. Each node delivers adds own *DSUID* to the summary, together with all *DSUID*s received from the downlinks. As the only nodes using the *DSUID* data are those at which the transition instances reside, they are the only ones at which the transition instance *DSUID*s need to be kept. Call the facility in which the transition instance *DSUID*s are stored a *DSUID table*. These tables contain the data about the transition instance *DSUID*s and is kept in synchrony with the *DSUID* labels delivered through the summaries and matches. Unrelated *DSUID* table instances are not guaranteed to be synchronized, for performance reasons. Synchronizing the *DSUID* tables would entail the need for distributed transactions. Omitting the synchronization leads to a system state in which different nodes may hold conflicting *DSUID* tables.

It can therefore occur that, for some contracting transition t there exist different transition instances, at nodes α and β, $t \diamond \alpha$ and $t \diamond \beta$ respectively, to which the *related* tokens from distinct places from $\circ t$ get delivered, although they should have been delivered to a single instance of t. These tokens trigger the guard of their respective transition[17] but as they are not present simultaneously at the same node, the transition t would never get executed. We therefore have two opposing forces in the token delivery: the delivery proximity rule, stating that for

[17]See Chapter 2 for the description of the guard in the blackboard-based CPN implementation.

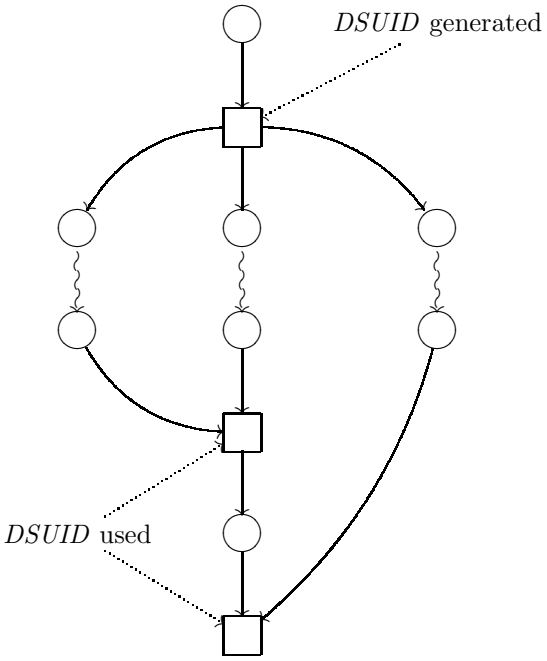


Figure 7.8: Brokering. The example shows an *expand* transition that generates *DSUID*s for three separate token streams. The streams are joined by two *contract* transitions further down the CPN.

execution a token must be delivered to some of the transition instances closest to it, and the *DSUID* table asynchrony, which prevents the related tokens to be always delivered to the same transition instance.

It necessarily means that the tokens will be misdelivered at runtime. The misdeliveries are to occur when the nodes join or leave the network, as these events change the set of the mentioned c minimum-distance nodes. Provided that the network shows eventual stability in terms of the available nodes, the misdelivery can be handled by requiring the secondary forwarding of the misdelivered objects. The criterion for misdelivery is based on comparing the actual *DSUID* of a token on the local blackboard[18]. This requires an addition to the token and summary/match reception algorithms, for additional inspection of each token for the contract transition instances. In a nutshell, each change to the producer/-consumer matches (propagating the consumer *DSUID*s) triggers a local token inspection on the local blackboard of the affected node. If, for some object o the updated match contains an *DSUID* which is closer to o than the local transition,

[18]The blackboard implementation of the token processing mechanism for CPN is given in Chapter 2 on page 34.

the object is re-forwarded to the owner of the respective *DSUID*.

7.5 Summary

In this Chapter we supplied to the DWEAM model the final ingredient which makes the execution of distributed workflows possible: the model for executing distributed programs within the Distributed Workflow Execution Architecture for Mobile (DWEAM) context.

We started off by introducing the location-independent mechanisms for object delivery. We argued that such mechanisms are preferable for dynamic networks in comparison to the delivery methods based on network-level addressing. The general objection to network-level addressing is that it bears no resemblance to the actual traffic patterns required in a distributed system, as the addresses reflect the *network structure*, whereas in fact it is the *data distribution structure* that needs to be reflected. Adopting content-based addressing alleviates this issue, as content-based addressing describes the desired objects in terms of the features that remain fairly constant during the system runtime, the data *semantics*. The content-based addressing approach, while common in database systems, seems to be fairly uncommon for the use in data delivery across system boundaries.

Curiously enough, content-based addressing is fairly common within the boundaries of a single system, even if the system itself consists of multiple computers joined together. To observe this claim as true one can look at database applications, where the objects of interest are routinely chosen based on the values of various fields. The field values are drawn from the tables, in case of a relational database, or objects, in case of an object-oriented database.

Unfortunately, the paradigm shift from content-based to network-based addresses comes at a price. Without the explicit notions of a *sender* and a *receiver* a whole flurry of activity needs to be performed for the service discovery and the matching. Hereby the notions of sender and receiver are replaced by the notions of producer and consumer. The change naturally reflects the shift from network-based to content-based addressing.

Although the content-based addressing schemes at first seem to be an unnecessary complication of an otherwise hygienic approach to distributed communication, it turns out that flexible, extensible, and scalable distributed applications do in fact require content-based addressing. However, the frameworks we were confronted with during the work on DWEAM (COUGAAR and Java Agent Development Environment (JADE), which we consider to be typical representatives of the mainstream, general purpose, multi-agent platforms). This is painfully realized to be true when implementing a distributed application and starting from the "near end" with simple use cases that disregard distributed problems, and realizing, halfway into the implementation, the scope of change needed in order to bring into play the full scope for transparency, failover, distribution etc. At the basis of all of this functionality is one or the other content-based addressing

scheme. Unfortunately, one is then left with the conlclusion that no standard *generic* scheme is available for content-based communication, and that it is the task of the application builder to invent an ad-hoc solution. For DWEAM we provided a generic approach to content-based data distribution, in hope to reinforce the view in support of the content-based approach.

For the location independent data distribution to work, a notion of a commonly understood *Dataspace* was necessary, and it had to be equipped with the operators for manipulating the data contained within. At this point a serious tradeoff had to be considered. The *Dataspace* had to, in essence, be a set encompassing all the known, but also all the yet *unknown* (but constructible) types that may arise in distributed systems. This requirement meant in effect that commonly recognized semantics of the regions of the *Dataspace* must exist. It also means that the slightly incompatible *Dataspace* regions can apear due to faulty or incremental *Dataspace* design, and that nothing but design discipline can help prevent the ambiguities arising from there. Unfortunately there seems little one can do to prevent (ab)using the *Dataspace* concept to make incorrect programs, in line of Brooks' observation that: "there is no silver bullet [which irrevocably solves non-incidental programming issues]" [16] . Once the *Dataspace* and the operations have been defined, we could specify the method used for matching the producers and consumers, relying on the availability of the CBT (see Chapter 6). The matching method relied on the fact that the CBT is a tree, and hence on the existence of a core node, to guarantee that matches between compatible nodes would eventually be found. A generic method for matching was given, as well as a specific method which involved the use of Bloom filtering to represent sets of character sequences, that can be used for tagging messages and efficient recovery of the appropriate content objects.

Finally the issues involved in the workflow mechanics were described, namely those arising from the fact that the bookkeeping must be held at the level of *object groups* or classes, rather than at the level of individual objects. We introduced the versioning method allowing for multiple object instances to reside in the system, as a consequence of multiple transition instances that produce them. This required the versioning of not only the objects, but also the summaries themselves. The final issue was the synchronized token delivery for contract transitions. This was resolved by introducing an unique identifier scheme for the contract transitions, and the distance scheme, whereby tokens are only delivered to closest nodes in the identifier space. Further, dependent tokens were assigned the same identifier, which is assigned early in the execution of the respective CPN. It was thereby ensured that the related tokens would ultimately be delivered to the same transition instance. The temporarily mismatched view on the available transition instances is solved by allowing nodes to forward misdelivered objects closer to their intended destinations.

Chapter 8

Conclusion

In this thesis, we looked in detail into the design of Distributed Workflow Execution Architecture for Mobile (DWEAM). In this final Chapter, we will summarize the results from the thesis, outline the possibilities for further work, as well as comment about the practical experience gained and possible further developments.

8.1 Introduction

To understand the design decisions made in the development of DWEAM one needs to understand the context within which it was developed. DWEAM was made under the aegis of the project Combined Systems. The project Combined Systems was set up to investigate the possibilities of introducing information systems in various phases of crisis management. One of the concerns within Combined Systems was the deployment of information systems in the field alongside emergency workers.

This concern presented a clear challenge: the requirement was to deploy a distributed system (consisting of PDAs devices carried by the workers) into a hostile environment, and entrust it with carrying out a computational task using a set of volatile computation devices (PDAs) and a volatile wireless interconnection. Such a requirement suggested that the operating environment for DWEAM plays a crucial role in the runtime, affecting both the computational nodes, and the connections between them. This is far from the clean-room environment in which the conventional distributed systems are running, where errors are infrequent and the interconnections do not change with time. Here, the structure of the interconnection as well as the availability of the nodes are not known in advance, and there is little initial structure to rely on.

To organize a meaningful computation in such a chaotic environment we needed to develop a set of unique techniques. These techniques are the thesis contributions that we mention in the next Section.

8.2 Why Distributed Workflow Execution Now

The work on DWEAM has for the most part been an endeavour on the path seldom taken in the world of distributed computing. From the literature surveys, it is seen that most of the research interest in distributed computing has been in tightly coupled distributed systems in which the computational resources are scarce. By tight coupling we are encompassing a set of implied operating conditions of the distributed systems, such as: the geographical co-location of the computers, the scarcity of the resources, the reliability and high bandwidth of the network etc.

There seemed to have been comparatively little interest in distributed computing with a large number of diverse, unreliable and unreliably connected devices. We argue, with reasonable confidence, that the reason for this is that the research in distributed computing has been motivated by fashionable applications. The applications of distributed computing have predominantly been either in the world of scientific computation, the business world etc. In these scenarios, what was required to get the job done was a computational powerhouse, with all its computational power concetrated at one physical site.

As the Internet became ever more popular among casual users, this computational powerhouse scenario surprisingly remained unchanged. In effect, little has fundamentally changed in the way we use the computer systems now, to what we used to do say a decade or two ago. Although, admittedly, the number of sites we are able to reach today from our workstations has grown immensely. Today, as before, we mostly use our computers as terminals to access some computational powerhouse for our own purpose. We use them for shopping, for searching, for file transfer. We are able to use them at a great physical distance, but the nature of the work being performed has remained centralized. In fact, only a few truly distributed applications are widely deployed today. These are almost exclusively file sharing applications, built on Bit Torrent and similar distributed transfer protocols. Further, scientific applications such as Boinc [13] appeared in recent years[1], but their data flow model is elementary: a chunk of work is received from a central server, processed and returned; the main application logic remained centralized.

During the work on DWEAM, we emphasized the completely decentralized solutions enabling the distributed execution of arbitrary workflows, not just the fixed ones as is the case with Boinc. The interest in implementing arbitrary workflows seems to have been proven justified, as Google Inc. wrote the first articles about MapReduce [24, 47] while the work on this thesis was still unfolding. MapReduce was produced from the massively parallel code used to power Google's search engine. Interestingly, MapReduce did not exist explicitly in the early years of Google's development. It has apparently been factored out of the existing code instead, as the generic method for easy parallelization. While of course there is little known how it is used, from what little information one can glean from

[1] Mid 2000's as of this writing.

MapReduce, there definitely exists a commercially lucrative future for distributed workflow execution. Google's success indicates this clearly, to our understanding.

8.3 Future Work

The work on DWEAM and distributed workflow execution motivates an interesting path for both research and applications. The algorithms for service discovery and work distribution methods may prove useful for non-volatile environments as well.

For fixed networks, the DWEAM-like implementations of service discovery, using proximity-based methods instead of content-based matching are being developed as a sequel to DWEAM since the end of the Combined project. In the proximity-based service discovery, the notion of *Dataspace* (see Chapter 7) still exists, in form similar to that from DWEAM. However, in the proximity-based approach, the *Dataspace* is set up as a metric space (with a metric tailor-made to fit the application), and queries into the *Dataspace* are performed by specifying ball-like *Dataspace* subsets by defining a point and a radius. Such a metric *Dataspace* is easily managed through a Distributed Hash Table (DHT) facility, for which a variety of algorithms are known (e.g. Chord [63], Tapestry [92]). The proximity-based queries substitute the fairly complicated content-based matching.

At the time of this writing, the metric *Dataspace* extension of the DWEAM mechanisms is in the specification phase. The aim is to make a system specification which, in a manner similar to that of MapReduce, enables one to execute distributed tasks by manipulating objects accessible through handles placed in the *Dataspace*.

In the sense of Brooks [16], this approach does not represent a silver bullet that single-handedly solves all the problems of distributed computing. Through the experience gained by working on DWEAM we are fairly convinced that such a device is not likely to exist. Rather unfortunately, the design of distributed systems seems to always have to be tweaked toward a specific application. As a consequence, this *Dataspace* approach seems to be tweaked towards the applications which are naturally expressed in form of workflows. We expect that among those that can be set up in the form of a workflow, numerous useful applications can be found.

* * *

The Eight Fallacies of Distributed Computing (by Peter Deutsch)

Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause big trouble and painful learning experiences.

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

Bibliography

[1] Combined web page. Online reference. http://combined.decis.nl.

[2] Distributed.Net Web Site. http://www.distributed.net.

[3] Linda in a mobile environment (LIME). Online reference. http://lime.sourceforge.net/.

[4] Wireless foundations. http://www.eecs.berkeley.edu/wireless/.

[5] *Semantic Data Modeling*. Prentice Hall, Englewood Cliffs, 1992.

[6] Europe's mobile market penetration is set to breach 100% in 2006 or early 2007. In *Business Wire*. Gale Group, November 2005. http://www.researchandmarkets.com/reports/c28137.

[7] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and interpretation of computer programs*. MIT Press, Cambridge, MA, USA, 1985.

[8] Marco Avvenuti, Alessio Vecchio, and Giovanni Turi. A cross-layer approach for publish/subscribe in mobile ad hoc networks. In Thomas Magedanz, Ahmed Karmouch, Samuel Pierre, and Iakovos S. Venieris, editors, *MATA*, volume 3744 of *Lecture Notes in Computer Science*, pages 203–214. Springer, 2005.

[9] Tony Ballardie, Paul Francis, and Jon Crowcroft. Core Based trees (CBT). In *SIGCOMM '93: Conference proceedings on Communications architectures, protocols and applications*, pages 85–95. ACM Press, 1993.

[10] BBN Technologies. *The Cougaar Architecture Guide*, 2004.

[11] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[12] Barry W. Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, Winsor A. Brown, Sunita Chulani, and Chris Abts. *Software Cost Estimation with Cocomo II (with CD-ROM)*. Prentice Hall PTR, New Jersey, January 2000.

[13] BOINC: Berkeley open infrastructure for network computing. Online reference. http://boinc.berkeley.edu/.

[14] J. P. Bowen. *Formal Specification and Documentation Using Z: A Case Study Approach*. International Thomson Computer Press, 1996.

[15] R. K. Brayton and F. Somenzi. An exact minimizer for boolean relations. In *IEEE International Conference on Computer-Aided Design*, pages 316–319. IEEE, November 1989.

[16] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley Professional, August 1995.

[17] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[18] Antonio Carzaniga and Alexander L. Wolf. Content-based networking: A new communication infrastructure. In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, number 2538 in Lecture Notes in Computer Science, pages 59–68, Scottsdale, Arizona, October 2001. Springer-Verlag.

[19] K. Mani Chandy and Jayadev Misra. The drinking philosopher's problem. *ACM Trans. Program. Lang. Syst.*, 6(4):632–646, 1984.

[20] Daniel Corkill. Blackboard Systems. *AI Expert*, 6(9), January 1991.

[21] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley Series in Telecommunications. John Wiley & Sons, New York, NY, USA, 1991.

[22] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[23] G. de Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw Hill, 1994.

[24] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplifed data processing on large clusters. *Operating Systems Design and Implementation*, pages 137–149, 2004.

[25] Reinhard Diestel. *Graph Theory*. Springer-Verlag New York, 2000.

[26] Edsger W. Dijkstra. Notes on Structured Programming. circulated privately, April 1970.

[27] Edsger W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Inf.*, 1:115–138, 1971.

[28] Filip Miletić. About Blackboards and Petri Nets. Technical Report TR-BBPN-05, Delft University of Technology, Circuits and Systems Group, Mekelweg 4, 2628CD Delft, The Netherlands, August 2005.

[29] Filip Miletić and Patrick Dewilde. Data Storage in Unreliable Multi-agent networks. In *Proceedings AAMAS*, July 2005. Utrecht.

[30] Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *Fundamentals of Computation Theory*, pages 127–140, 1983.

[31] Folding at home. Online reference. http://folding.stanford.edu.

[32] Freenet. Online reference. http://freenet.sf.net/.

[33] Felix Gaertner. Revisiting Liveness Properties in the Context of Secure Systems. Technical report, 2002.

[34] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

[35] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[36] Gnutella. Online reference. http://gnutella.wego.com/.

[37] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics; Second Edition*. Addison-Wesley, 1997. GRA r 94:1 1.Ex.

[38] Gryphon event notification. Online reference. http://www.research.ibm.com/gryphon.

[39] P. Gupta and P. Kumar. Capacity of wireless networks. Technical report, University of Illinois, Urbana-Champaign, 1999.

[40] Charles Anthony Richard Hoare. Communicating sequential processes. *Commun. ACM*, 26(1):100–106, 1983.

[41] P. Homburg, M. van Steen, and A. S. Tanenbaum. An architecture for a wide area distributed system. In *Seventh ACM SIGOPS European Workshop*, pages 75–82, Connemara, Ireland, 1996.

[42] Yongqiang Huang and Hector Garcia-Molina. Publish/subscribe tree construction in wireless ad-hoc networks. In *MDM '03: Proceedings of the 4th International Conference on Mobile Data Management*, pages 122–140. Springer-Verlag, 2003.

[43] John Turek and Dennis Shasha. The Many Faces of Consensus in Distributed Systems. *Computer*, 25(6):8–17, 1992.

[44] T. Johnson. Designing a distributed queue. In *SPDP '95: Proceedings of the 7th IEEE Symposium on Parallel and Distributeed Processing*, page 304, Washington, DC, USA, 1995. IEEE Computer Society.

[45] U. C. Kozat and L. Tassiulas. Service discovery in mobile ad hoc networks: an overall perspective on architectural choices and network layer support issues. *Ad Hoc Networks*, 2(1):23–44, January 2004.

[46] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[47] Ralf Lämmel. Google's MapReduce Programming Model – Revisited. Draft; Online since 2 January, 2006; 26 pages, 22 January 2006.

[48] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3(2):125–143, 1977.

[49] Vincent Lenders, Martin May, and Bernhard Plattner. Service discovery in mobile ad hoc networks: A field theoretic approach. In *WOWMOM '05: Proceedings of the Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM'05)*, pages 120–130, Washington, DC, USA, 2005. IEEE Computer Society.

[50] Michael Luck, Peter McBurney, and Chris Preist. *Agent Technology: Enabling Next Generation Computing*. The Agentlink Community, 2002.

[51] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996. LYN n 96:1 P-Ex.

[52] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996. LYN n 96:1 P-Ex, Theorem 3.1, pp. 27.

[53] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric, 2002.

[54] R. Meester and R. Roy. *Continuum Percolation*. Number 119 in Cambridge Tracts in Mathematics. Cambridge University Press, 1996.

[55] Sun Microsystems. Jini network technology. http://www.jini.org.

[56] Filip Miletić and Patrick Dewilde. Distributed coding in multiagent systems. In *IEEE Conference on Systems, Man and Cybernetics*. IEEE, October 2004.

[57] Filip Miletić and Patrick Dewilde. Coding approach to fault tolerance in multi-agent systems. In *IEEE Conference on Knowledge Intensive Multiagent Systems*. IEEE, April 2005.

[58] Filip Miletic and Patrick Dewilde. Data storage in unreliable multi-agent networks. In Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh, and Michael Wooldridge, editors, *AAMAS*, pages 1339–1340. ACM, 2005.

[59] Filip Miletic and Patrick Dewilde. Design considerations for an infrastructure-less mobile middleware platform. In Katja Verbeeck, Karl Tuyls, Ann Nowé, Bernard Manderick, and Bart Kuijpers, editors, *BNAIC*, pages 174–179. Koninklijke Vlaamse Academie van Belgie voor Wetenschappen en Kunsten, 2005.

[60] Filip Miletic and Patrick Dewilde. A distributed structure for service description forwarding in mobile multi-agent systems. *Intl. Tran. Systems Science and Applications*, 2(3):227–244, 2006.

[61] M. Mitzenmacher. Compressed bloom filters. In *Proc. of the 20th Annual ACM Symposium on Principles of Distributed Computing*, IEEE/ACM Trans. on Networking, pages 144–150, 2001.

[62] Mojonation. Online reference. http://www.mojonation.net/.

[63] Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM 2001*, San Diego, USA, September 2001.

[64] Napster. Online reference. http://www.napster.com/.

[65] Paal Engelstad and Yan Zheng. Evaluation of Service Discovery Architectures for Mobile Ad Hoc Networks. In *WONS*, pages 2–15. IEEE Computer Society, 2005.

[66] G. Picco, G. Cugola, and A. Murphy. Efficient content-based event dispatching in the presence of topological reconfigurations. In *Proc. of the 23 Int. Conf. on Distributed Computing Systems (ICDCS 2003)*, 2003.

[67] Amir Qayyum, Laurent Viennot, and Anis Laouiti. Multipoint relaying: An efficient technique for flooding in mobile wireless networks. Technical Report Research Report RR-3898, INRIA, February 2000.

[68] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. Technical Report TR-00-010, University of California at Berkeley, Berkeley, CA, 2000.

[69] Wolfgang Reisig. *Elements of Distributed Algorithms*. Springer, 1998.

[70] T. Richardson and R. Urbanke. Modern coding theory, June 2003.

[71] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–351, 2001.

[72] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1996.

[73] Francoise Sailhan and Valerie Issarny. Scalable service discovery for manet. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 235–244, Washington, DC, USA, 2005. IEEE Computer Society.

[74] Seti at home. Online reference. http://setiathome.ssl.berkeley.edu/.

[75] Claude E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27(3):379–423, July 1948. Continued 27(4):623-656, October 1948.

[76] G. Smith. *An Object-Oriented Approach to Formal Specification*. PhD thesis, St. Lucia 4072, Australia, 1992.

[77] Graeme Smith. A semantic integration of object-Z and CSP for the specification of concurrent systems. In John Fitzgerald, Cliff B. Jones, and Peter Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997)*, volume 1313, pages 62–81. Springer-Verlag, 1997.

[78] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science, 2nd edition, 1992.

[79] V. S. Subrahmanian, Piero Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Ozcan, and Robert Ross. *Heterogeneous Agent Systems*. MIT Press/AAAI Press, Cambridge, MA, USA, 2000.

[80] V. S. Subrahmanian, Piero Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Ozcan, and Robert Ross. *Heterogeneous Agent Systems*, chapter 1, page 21. MIT Press/AAAI Press, Cambridge, MA, USA, 2000.

[81] M. Turoff, M. Chumer, B. van de Walle, and X. Yao. The design of a dynamic emergency response management information system. *Journal of Information Technology Theory and Applications*, 5:4:1–36, 2004.

[82] Chris J. van Aart, Bob Wielinga, and Guus Schreiber. Organizational building blocks for design of distributed intelligent system. *Int. J. Hum.-Comput. Stud.*, 61(5):567–599, 2004.

[83] A. J. van der Hoeven, A. A. de Lange, E. F. Deprettere, and P. M. Dewilde. A new model for the high level description and simulation of vlsi networks. In *DAC '89: Proceedings of the 26th ACM/IEEE conference on Design automation*, pages 738–741, New York, NY, USA, 1989. ACM Press.

[84] Pieter van der Wolf. *Architecture of an Open and Efficient CAD Framework*. PhD thesis, TU Delft, June 1993.

[85] G. Vitaglione, F. Quarta, and E. Cortese. Scalability and performance of jade message transport system, 2002.

[86] Y. Watanabe and R. K. Brayton. Heuristic minimization of multiple-valued relations. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 12(10):1458–1472, October 1993.

[87] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison.

[88] David A. Wheeler. SLOCCount user's guide. Online reference, http://www.dwheeler.com/sloccount/sloccount.html, August 2004. version 2.26.

[89] Berkeley wireless foundations. Online reference. http://www.eecs.berkeley.edu/wireless/vision.html.

[90] Workflow. online reference. http://en.wikipedia.org/Workflow.

[91] Yi-Min Wang, Lili Liu, Chad Verbowski, Dimitris Achlioptas, Gautam Das, and Paul Larson. Summary-based Routing for Content-based Event Distribution Networks. *ACM SIGCOMM Computer Communications Review*, 34(5):59–74, October 2004.

[92] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, January 2004.

Acknowledgments

This thesis is an outcome of my Ph. D. research done at the Circuits and Systems (CAS) group of Delft University of Technology. The research was conducted from January 2003 until January 2007. During that time, numerous people influenced constructively on its making. I am taking the opportunity to thank them here.

First and foremost, my gratitude goes to prof. dr. ir. Patrick Dewilde. In late 2002, he gave me the opportunity to become the first in the CAS group on the Intelligent Systems research track. His expert guidance has been invaluable throughout the research ever since.

Further thanks go to all the members of the CAS group. They created just the right hue of a relaxed and productive work environment that I had the privilege of enjoying in the past years. They welcomed me, helped on numerous occasions with issues simple and complex alike, asked pointed questions, engaged into discussions, and in short did everything to make the time spent at CAS worth the while. I hope they would continue in the same spirit in the upcoming years.

No thank-you note would be complete without the mention of my fellow researchers at DECIS Lab in Delft. Thanks to dr. Kees Nieuwenhuis, without whom DECIS would not be the same. Also to Paul Burghardt, the project leader of Combined and an expert cat-herder who balanced the adventurous spirit of the researchers with the necessity of producing practical, usable results. The quick tip o' the hat does no justice to you, but is second best to writing a book in itself on all the ways that you have influenced me, guys and gals: Aletta, Bernard, Bogdan, Chris, Els, Geert, Gregor, Guido, Hakan ("the Hackman"), Jan (×2), Jan Maarten, Jeroen, Josine, Joost, Kees, Lauwrens, Leo, Leon, Martijn, Martien, Marielle, Marinus, Niek, Patrick (×2), Renee, Rianne, Peter, Ronald, Siska, Stijn, Thomas, and Wim.

Thanks to Kientz-sensei and the members of Aikido Stichting Delft (ASD) for the best break from the "all work, no play" routine.

Special thanks go to Milan, Tanja, Ognjen and Maša for all their support, for being the family away from home. Also Marc, who has taken on so many roles in my life it becomes difficult to explain them all. He has been a friend, adviser, supporter, handyman, language tutor, drink buddy, fellow adventurer, critic and more. As if that were not enough, I also had him proof-read and translate the thesis propositions. The Djapić-Litavski family gave extensive advice on the ins

and outs, ups and downs, lefts and rights, backs and forths of expat life, always with a freshly brewed cup of latest news, never forgetting to invite me to their trips around Dutch countryside.

Thank you, friends and family worldwide, for being there in need: Aleksa, Ana, Bežanija, Boba, Bondža, Boris, Coa Pop, Darko, Djenka, Dušica, Edin, Fića, Habi, Jasmina, Joe, Jomu, Jovana, Klinceza, Laza, Leka, Maja, Marina, Marko, Mighty, Milan, Miloš, Nada, Nataša, Pedja, Sava, Silva, Sloba, Sofija, Vlada, Zmaj, and Željko.

Most of all, thanks to my parents, Slavica and Dragiša. And to Marija, for her continued support throughout the years when I needed it the most.

Filip Miletić
April 2007
Arcen

Samenvatting

Een architectuur voor executie van taken onder ongunstige omstandigheden
(An Architecture for Task Execution in Adverse Environments)
Filip Miletić

Dit proefschrift beschrijft een architectuur voor gedistribueerd rekenen in mobiele omgevingen. Het begrip *workflow* staat voor het operationele aspect van een werk procedure. Het begrip omvat de structuur van de taken, wie ze uitvoert, wat hun operationele wijze is, hoe zij gesynchronizeerd worden, hoe de informatie vloeit om de taken te ondersteunen en hoe zij worden bijgehouden in het systeem. De ontworpen architectuur is getest in een prototype implementatie genaamd 'Distributed Workflow Execution Architecture for Mobile (DWEAM)'.

Interesse in architecturen voor gedistribueerde executie van 'workflows' bestaat sinds lang. Er is echter nog nooit een adequate behandeling van 'workflow' executie gegeven voor mobiele systemen. Een mobiel 'workflow' systeem kan van groot belang zijn voor gebruikers die een gecoördineerde opdracht moeten uitvoeren in een complexe omgeving en zelf moeten instaan voor het tot stand komen van de coördinatie. Typische gebruikers zijn leden van een ploeg die optreedt bij de bestrijding van een ramp, bijvoorbeeld een ploeg brandweerlui, politieagenten of medisch personeel, waarbij de samenwerking tussen de leden gehinderd wordt door mobiliteit en een slechte communicatieomgeving. Communicatie met een GSM netwerk, dikwijls gebruikt in een stadsomgeving, levert dikwijls niet de gewenste 'Quality of Service', als de beschikbaarheid ervan al niet verstoord wordt door de zich uitbreidende ramp, schade of overbelasting van de infrastructuur. Ouderwetse, toegespitste systemen zoals Walkie-Talkie omgevingen zijn ontworpen voor communicatie tussen mensen alleen en bieden geen verbinding met een informatieverstrekkende achtergrond. Om deze redenen hebben wij onze inspanningen toegespitst op een systeemarchitectuur die tegelijk communicatie en onafhankelijkheid van de infrastructuur nastreeft, èn in staat is om zowel de werkprocedures voor menselijke agenten als de informatievoorziening te ondersteunen.

In dit proefschrift beginnen we met het DWEAM probleem in de bredere context van systemen met meerdere initiatieven, actoren en agenten te plaatsen.

Daaruit blijkt dat DWEAM slechts een componente is van een breder systeem dat de naam Chaotic Open-World Multi-Agent Based Intelligent Networked Decision Support System (Combined) gekregen heeft. We geven een gestroomlijnde beschrijving van Combined en de eisen gesteld aan het gecombineerde systeem. Volgend op die beschrijving geven we een overzicht van, en commentaar op gerelateerd werk uit de literatuur. We beschrijven de 'toolkit' dat we zullen gebruiken in de overige hoofdstukken, met name de 'Object-Z taal' en de 'Coloured Petri Nets' (Coloured Petri Net (CPN)). We sluiten dit eerste hoofdstuk af met een formele beschrijving van de taken die DWEAM moet vervullen en de beschrijving van het gedistribueerde 'blackboard' dat als achtergrondskader gebruikt wordt.

We vervolgen met een beschrijving van de methode die we gebruiken om de informele, onderling verbonden taakbeschrijvingen om te zetten in een CPN. Het systeem vertaalt vervolgens de CPN beschrijving in een implementatie die gebruik maakt van het gedistribueerde 'blackboard'.

In een volgend hoofdstuk geven we de beschrijving van de operationele omgeving en het geheugen model. We onderzoeken de connectiviteit van een verzameling knooppunten in twee dimensies. Vervolgens komt dan de geheugen opslag aan de orde, we schatten de 'performance' van de data partitionering en hoe preservering van de 'tokens' kan gewaarborgd worden.

Wat de data distributie betreft, komen de noties 'producent' en 'gebruiker' aan de orde. We definiëren de Service Discovery Problem (SDP). Daarna wordt de CBT gebruikt om een oplossing te geven voor de SDP, door producenten en gebruikers te vinden die compatiebel zijn, namelijk die data met elkaar willen of moeten uitwisselen. We geven een gedetailleerde analyse van het algoritme dat de CBT opbouwt, en gebruiken daarbij de CPN beschrijving van de diverse fasen van de operatie. Ook de 'performance' komt hierbij aan de orde.

Al de data in het DWEAM systeem moet voldoen aan het *Dataspace* model. Dit laat toe een 'Matching Algorithm' te construeren. De CBT structuur die voorheen is opgebouwd wordt gebruikt om de 'matching' tussen compatiebele producenten en gebruikers te realizeren. We geven een bewijs voor het 'matching algorithm' en een CPN beschrijving van zijn implementatie.

We sluiten de thesis af met een lijst contributies, beschouwingen over het perspectief van gedistribueerde 'workflow' systemen in een hedendaagse context en ideeën voor verder werk in deze context.

About the Author

Filip Miletić was born on November 22, 1978 in Kruševac, Serbia. In 1997, he graduated from the secondary school "Gimnazija" ("Gymnasium") in Kruševac. The same year, he entered the School of Electrical Engineering, at the University of Belgrade, in Serbia. During the studies, he was on an internship at the Circuits and Systems Group of Delft University of Technology, from October 2001 until April 2002. There he developed components for the Leon SPARC soft-processor. He then returned to Belgrade to finish his studies.

In 2002, he obtained the title Graduate Electrical Engineer from the University of Belgrade, also on the topic of soft-processor development. In 2003, he started his Ph. D. studies at the Circuits and Systems Group in the context of intelligent systems design. He participated in the project Combined, investigating multi-agent approaches for building crisis-management information systems. Starting January 2007, he works at Océ Technologies B. V. in Venlo, Netherlands as a hardware engineer.

Propositions accompanying the thesis
An Architecture for Task Execution in Adverse Environments
by Filip Miletić

1. Content-based addressing is preferable for dynamic networks in comparison to the usual end-to-end addressing as used on the Internet today. The end-to-end addressing bears no resemblance to the traffic patterns, as the addresses reflect the network structure, and are invalidated due to changes in the network. [This thesis]

2. When Internet was first created, it was designed to allow an arbitrary *pair* of users to communicate via connected computers. Further progress of the Internet must rely on decentralized groupware.

3. Internet-wide groupware must balance its merits with the ease of use. Internet Relay Chat (IRC) and *Usenet*, services with inherent many-to-many communication, lost in popularity despite their potential in favor of centralized services such as the World-Wide Web (WWW) and Instant Messaging (IM), for not striking a good balance.

4. The service guarantees of the Internet protocols (TCP and UDP) are oft misunderstood in the computer engineering community. UDP is considered "unreliable," and TCP "reliable," following the original TCP/IP glossary. But this is wrong, for "reliable communication" is an oxymoron. Still, assuming TCP reliable even makes it into software designs, yielding programs that lock up when their TCP connections fail.

5. If the best invention in the world is the sliced bread, then the toaster must be the second best. That is, all good ideas need the right environment to bake in.

6. The C compilers are notorious for being unhelpful with program diagnostics. In your daily use of the C compiler, two error messages appear much more frequently than any other. They are: "Parse error" and "Segmentation fault." These two translate into less technical language as: "There is an error somewhere in your code," and "I told you so."

7. One's command of a foreign language correlates well with one's ability to amuse another in that language.

8. Proving impossibility theorems is the best job for a computer scientist. Do it right, and you do not have to write even a single line of code, because you have already proved that writing them leads nowhere.

9. It is easier to move a networking problem around (for example, by moving the problem to a different part of the overall network architecture) than it is to solve it. **Corollary:** It is always possible to add another level of indirection. [Truth #6, from "*RFC1925: The Twelve Networking Truths*"] **Added Corollary:** Network reliability issues set behind the horizon.

10. For workflow execution in dynamic networks, the traditional sender and receiver should be replaced by the notions of producer and consumer. Otherwise, the cost of service discovery becomes too high. [This thesis]

These propositions are considered opposable and defendable and as such have been approved by the supervisor, prof. dr. ir. Patrick M. Dewilde.

Stellingen behorend bij het proefschrift
An Architecture for Task Execution in Adverse Environments
door Filip Miletić

1. Op inhoud gebaseerde adressering heeft de voorkeur voor dynamische netwerken boven de end-to-end adressering zoals die tegenwoordig op Internet gebruikt wordt. De end-to-end adressering, lijkt niet op de verkeerspatronen, omdat de adressen de netwerk structuur weergeven en worden ongeldig gemaakt door veranderingen in het netwerk. [Dit proefschrift]

2. Toen Internet net bestond, was het ontworpen om twee gebruikers te laten communiceren via onder elkaar verbonden computers. Verdere ontwikkeling van het Internet moet vertrouwen op decentraal groupware.

3. Groupware voor Internet moet een balans vinden tussen zijn verdienste en gebruikersgemak. Internet Relay Chat en Usenet, diensten met inherente many-to-many communicatie, verloren populariteit ondanks hun mogelijkheden ten gunste van gecentraliseerde diensten als het World-Wide-Web en Instant Messaging, omdat ze de goede balans niet vonden.

4. De gegarandeerde diensten van Internet protocollen (TCP en UDP) worden vaak niet goed begrepen binnen de computer ontwikkel gemeenschap. UDP wordt als "onbetrouwbaar" beschouwd en TCP als "betrouwbaar", volgens de originele TCP/IP glossarium. Maar dit is verkeerd omdat onbetrouwbare communicatie een oxymoron is. Maar aangenomen dat TCP betrouwbaar is komt het zelfs in software designs voor, waardoor programma's vastlopen als de TCP verbinding wordt verbroken.

5. Als de beste uitvinding ter wereld gesneden brood is, dan is de broodrooster de op één na beste. Alle goede ideën hebben de juiste omgeving nodig om te roosteren.

6. C-compilers zijn berucht om hun slechte hulp bij programma diagnose. In het dagelijks gebruik van de C-compiler komen twee foutmeldingen veel vaker voor dan andere. Dat zijn: "Parse error" en "Segmentation fault". In minder technische termen betekenen deze meldingen: "Er zit ergens een fout in je code" en "Ik heb het toch gezegd?"

7. Iemands beheersing van een vreemde taal correleert goed met iemands vermogen om een ander te amuseren in die taal.

8. Het bewijzen van onmogelijkheids theorema's is de beste baan voor een computer wetenschapper. Als je het goed doet hoef je nooit een regel code te schrijven, omdat je al bewezen hebt dat het schrijven nergens leidt.

9. Het is makkelijker om een netwerk probleem door te schuiven (bijvoorbeeld door het probleem te verplaatsen naar een ander deel van de totale netwerk architectuur) dan om het op te lossen. Gevolg: Het is altijd mogelijk om een extra laag van indirectheid toe te voegen. Netwerk betrouwbaarheid kwesties over de horizon worden vooruitgeschoven.

10. Bij werkwijze uitvoering in dynamische netwerken, zouden de begrippen van verzender en ontvanger vervangen moeten worden door de begrippen van leverancier en gebruiker. Anders worden de kosten van de service discovery te hoog. [Dit proefschrift]

Deze stellingen worden opponeerbaar en verdedigbaar geacht en zijn als zodanig goedgekeurd door de promotor, prof. dr. ir. Patrick M. Dewilde.