NOTE TO USERS

Page(s) not included in the original manuscript and are unavailable from the author or university. The manuscript was scanned as received.

iv-v

This reproduction is the best copy available.

uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Andrew Roczniak

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Ph.D. (Electrical Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

On Robustness of Service-based Applications

TITRE DE LA THÈSE / TITLE OF THESIS

Abdulmotaleb El Saddik

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

Pierre Lévy

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Azzedine Boukerche

Dorina Petriu

Qusay Mahmoud

Emil Petriu

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

On Robustness of Service-based Applications

by

Andrew Roczniak

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the Ph.D. degree in
Electrical and Computer Engineering

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-48419-7
Our file Notre référence
ISBN: 978-0-494-48419-7

Abstract

Service-based applications is the name given to a situation arising when a user or a group of users interact with a set of distributed and autonomous functionalities such that some value is created for those users. For interoperability purposes, this interaction should take place through open and standardized interfaces.

Collaborative applications allow users to participate in achieving a common goal or objective. From a basic collaborative application that allows participants to author a multimedia document, three distinct functionalities were extracted and implemented as distributed and autonomous services. An example on how a group of users might interact with those services using standard set of protocols and off-the-shelf techniques is presented.

From the point of view of a user, a service-based application can only be useful if it provides some indication about its robustness. Creation of such applications therefore must consider the issue of undependability of a required service. A particular source of service failure (service provider refuses a service) or degraded performance (occasional lack of service or degraded quality of service) can be attributed to producers self-interested behavior.

The state of the art in dealing with the effects of rational behavior comprises trust and reputation networks, and methods based on game theory and mechanism design. These are presented and discussed in a broader survey of available strategies, tools and available techniques to analyze and mitigate effects of rational behavior.

The basis for the investigated solution is the observation that part of a service may be redundant with respect to some part of another service. Instead of adding redundancy to the system, our solution exploits the already existing redundancy to improve robustness of these applications.

The interaction with services is modeled as finite-state transducers and a heuristic is proposed to obtain redundancy between any pair of services for which an interaction model exists. Then, an algorithm that uses this inter-service redundancy to recreate the interaction with one service (target) from the other (candidate) is described. This approach allows users to have access to some specific functionality but not receive it from a service which may be deemed problematic. A qualitative reference framework to classify available services into candidate and target services respectively is also discussed.

The computational cost is polynomial with respect to services size, and in practice, the non-redundant functionality and related control need to be implemented locally.

Acknowledgements

Graduate studies are a bit like a journey through an unknown country. It can be difficult at times, but it is ultimately a deeply rewarding and enriching experience. I am indebted to my advisors, Dr. Abdulmotaleb El Saddik and Dr. Pierre Lévy for the opportunity to undertake this journey, for their guidance and their support. They were instrumental in guiding me to my destination while encouraging me to take roads less traveled but more scenic.

Long journeys are usually made after a suitable preparation. For help with this, I am indebted to Dr. Emil Petriu and Dr. George Costache.

I am grateful to my wife, my parents and my sister for their unwavering support and encouragements. I could not have done it without you!

Finally, I'd like to thank my fellow travelers for their encouragements, tips and enlightening conversations: Shawn Ferguson, Darek Baingo, Christian Champenois, Dima Kabrelyan, Max Nozin, Nick Soveiko, Daniel Shteyn, Cyril Wisniewski, Jonathan Layes and Jeff Fiedorowicz, you made my journey that much more interesting!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Broadband connectivity is arguably one of the major drivers of innovation in delivery methods of products and services to users. Preparing and filing your taxes, listening to radio stations or watching video clips and movies over the Internet are some of the application examples. The delivery of those services is usually based on a centralized network architecture (client/server) and rely on application-specific protocols and interfaces. Recent developments in decentralized network architecture such as peer-to-peer networks (P2P) and open standards as required by web services (WS) and service-oriented architectures (SOA), are creating however even more opportunities for innovation in delivery methods.

Although there is no universal agreement on the definition of P2P, one can argue that P2P is a network architecture whereby each entity can be a client as well as a server. Since there is no centralized control, such networks are easily created and operated, lowering barriers to introduction and delivery of new and compelling products. Such 'democratization' of networking is perhaps one of the most attractive characteristic of peer-to-peer architecture.

Web Services are based on standard protocols, which specify how a service should be described, found, accessed and used, irrespective of the specific platform or implementation that supports and instantiates that service. Service-oriented architectures leverage this standardization to create composite services, reuse existing functionality, integrate disparate software applications and potentially reduce complexity of large software systems. This variety of desired outcomes or goals is made possible by the fact

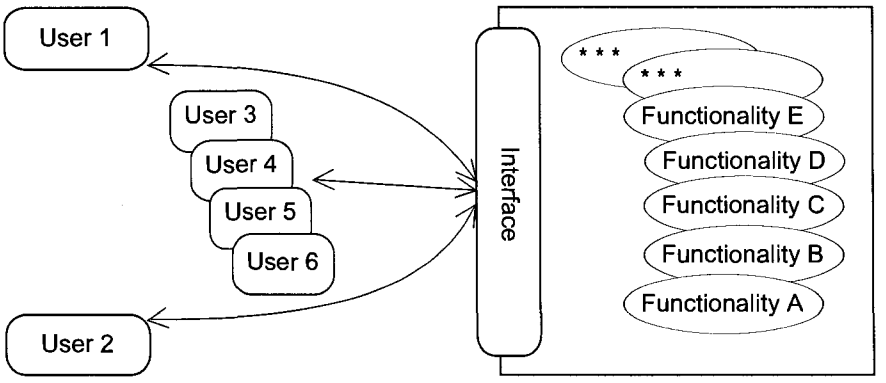

Figure 1.1: Conceptual view of a client-server application architecture

the standardization effort brought about by WS has created an application-level "lingua franca" for inter-networked machines.

Applications are thus built on top of client/server and emerging peer-to-peer architectures, and it can be argued that service-oriented architectures can similarly be used as a basis. If we think of an application as the result of the interaction between various functionalities, then it is helpful to see how those functionalities are allocated to different entities depending on which network architecture is used. In the context of client/server, all the functionalities would be co-located on one entity from the point of view of the clients (even though individual functionalities could interact between themselves through for example remote procedure call (RPC), remote method invocation (RMI) or common object request broker architecture (CORBA), or indeed web services).

In the P2P context on the other hand, functionalities are allocated to all participants[1] which in effect limits the number of supported functionalities and hence the number of supported applications. P2P-based applications thus tend to be designed for a specific purpose (help locating specific user or content, share storage space or bandwidth for example), unless a general framework such as JXTA[2] is used which supports creation and advertisement of new functionalities to participants on the network.

The middle ground, when functionalities are not necessarily allocated to one or all entities, can be supported by service-oriented architectures. From the point of view of the client then, it has the choice how the functionalities should be allocated. The specific

[1]This of course depends on the exact definition of P2P. It is consistent however with the view of an entity being able to function as a client as well as a server

[2]www.jxta.org

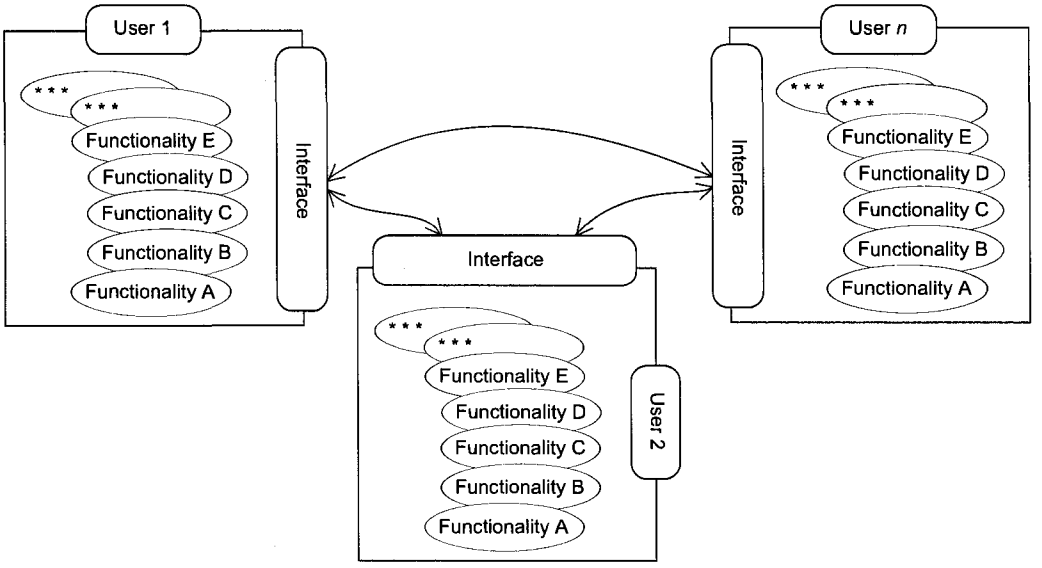

Figure 1.2: Conceptual view of a peer-to-peer application architecture

allocation is the result either of the selection of optimal functionalities or of the desired (de)centralization of functionalities.

Based on the above discussion, we can divide the application architecture into three classes; client/server (shown in Figure 1.1), P2P (shown in Figure 1.2) and service-based (shown in Figure 1.3). Benefits of each application architecture are shown in Table 1.1. The rest of the thesis is concerned with the service-based architecture of Figure 1.3.

Architecture	**Benefits**
Client-Server	Accountability
Peer-to-peer	Low barriers to entry
Service-based	Choice

Table 1.1: Benefits of an application architecture from users' point of view

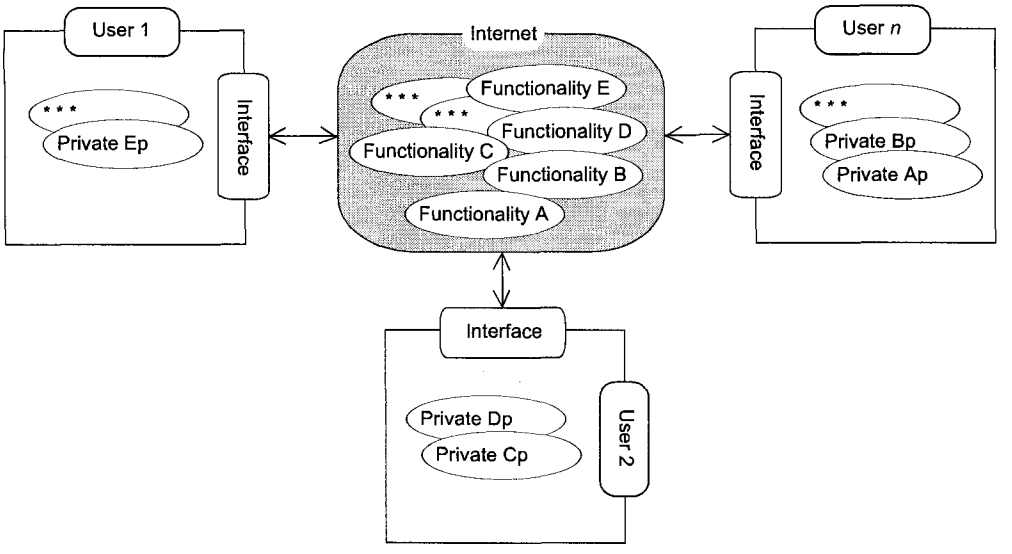

Figure 1.3: Conceptual view of a services-based application architecture

1.2 Motivation and Problem Description

There are many issues associated with the service-based applications. The requirements to find, describe and access services in a standard way create challenges in their own right, while still leaving the problem of how exactly to select services and have them interact in such a way as to achieve a coherent application.

The present thesis focuses on the problem created if one assumes that participants offering services can behave in a rational manner. One of the characteristics of service providers is that they may be under independent control in respect to both other service providers and to clients consuming its services. The practical implication of this independence is that the service providers decide when to actually provide a service and what its performance[3] will be. Effects of rational behavior is both a current and significant issue as evidenced by the wealth of recent research work.

As authors in [134] note, users can modify protocols or algorithms for self-interested reasons, and mention P2P search and routing and distributed auctions as some affected application domains. The effects of participants' rational behavior on file sharing in P2P environments are well documented [7, 13, 25, 60, 82, 87, 113, 124], and many solutions

[3]note that we assume that the service delivery channel does not affect performance, e.g. network bandwith is more than sufficient to deliver the service

to mitigate the unpredictable or undesired effects of rational behavior are proposed in applications as diverse as storage [35, 146, 104], service exchange [58], media streaming [63, 109], sharing [104, 92] and content distribution [40]. Additionally, many studies comparing incentive mechanisms exist [32, 14, 39, 52, 66, 70, 85, 86, 108, 149].

The effect of rational behavior on service-based applications can range from rendering the application broken (service provider refuses a service) to making the application undependable (occasional lack of service or degraded quality of service). From the point of view of the user, then, the question of robustness (and hence usefulness) of the service-based applications arises.

If an application depends on some service provider, then clearly it is sensitive to its availability and reliability. Over-engineering and building redundant systems is the usual method of mitigating the impact of uncertainty. The motivation for the approach investigated in this thesis is based on the observation that since services are supposed to be re-usable and independent of user's context, then there may be some redundancy between those services. Instead of adding redundancy therefore, the hope is to exploit the already existing redundancy to improve the robustness of service-based applications.

1.3 Model Description

The intent is to build a system from autonomous sub-systems. Autonomy implies that the sub-system controls both its quantitative and qualitative characteristics. In the present case, the system is an application and the sub-systems are service providers. Service providers control if and when the service is provided, and the quality of that service. Obviously, the user's perception of the quality of the application directly depends on service providers' characteristics.

The application and the service providers can generally be described in terms of their respective availability and reliability where, availability is the probability of an item to be in a state to perform a required function during a specified period of time and, reliability is the probability that an item will perform its intended function during a specified period of time. The interpretation of availability and reliability however is understood differently in the case of the service providers and the application.

The availability is usually expressed in terms of the expectation of uptime, mean time between failures (MTBF) and the expected downtime, mean time to repair (MTTR). A simple relationship between availability and MTBF and MTTR is $a = \frac{MTBF}{MTBF+MTTR}$. In the case of service providers, we have to take into account whether the infrastructure

supporting the service provider is functioning properly and is reachable over the Internet. On the other hand, the availability of the application can be a measure of how efficiently it can handle the task of finding and replacing unavailable service providers. Although availability is a very interesting topic in itself, it will not be the focus of this thesis.

To discuss reliability from the point of view of the service providers' and the application, it is helpful to first describe in abstract terms what a service is in the context of the model of a service provider.

A service is usually understood to be some functionality that is packaged and offered in a specific way. Services are self contained and operate as black boxes; all necessary components to accomplish a specific functionality are encapsulated within the service, allowing reuse in various contexts. Services expose their functionality through interfaces, and the interfaces are described and invoked in an open and standardized manner. Definitions supporting this view are given in [90, 110]. Services in the present context are additionally thought of as an algorithm taken from a set of equivalent algorithms where each member can be described in terms of both the level of performance achieved and the resources required to achieve that performance - its cost to the provider. This description allows each algorithm to be ranked relative to other members of that set. If we make the reasonable assumptions that the cost is not trivial and increases with increasing performance, then the obvious question is how does a service provider select a particular algorithm from a given set. For example, representing a service with performance/cost characteristics as s_{cost}^{perf} and the service provider as a set of equivalent algorithms $S_1 = \{s_{high}^{high}, s_{medium}^{medium}, s_{low}^{low}\}$, then from the user's point of view, the choice of a specific algorithm, either of which will accomplish the advertised function, is nondeterministic. Indeed, the user might not even be aware of the membership of the set.

Autonomy does not imply that service providers will behave rationally; however it certainly allows them to be self-interested, in the sense that they will choose a course of action that leads to their preferred outcome. If, for example, a service provider wants to offer as much service as possible at the lowest cost to itself, it can consistently advertise itself with the highest performance characteristics (assuming users seek highest performance) while always offering the algorithm that minimizes its cost.

The application (or any other consumer) does not have direct control over this selection, but may have some influence through some kind of incentive mechanism. Irrespective of the mechanism used, the consumer will not be absolutely guaranteed a specific algorithm (unless the mechanism can be proved to have certain properties[4]), and only

[4]Briefly described in the section 2.1.3

probabilistic assurance can be given.

If we quantify 'intended function' from the above definition of reliability into a set of relevant performance criteria, then reliability of a service provider can be expressed in terms of the probability of success in satisfying those criteria. The reliability of the service providers can thus be expressed in terms of the probability of obtaining a certain level of performance from the services they provide. The application's reliability - or robustness - in turn, can be expressed in terms of the reliability of the underlying service providers.

The following assumptions are made and will be explicitly integrated into the model:

1. All required services are available. This implies that those services were already "found", are reachable and are functioning properly
2. User selects what constitutes a failure in a service
3. An application fails if and only if at least one service it uses fails

1.3.1 Generating an Application from Services

A service is denoted as s (formal definition of service is deferred until section 5.2) and we assume that the execution (or consumption) of that service can be represented as $\text{EX}(s)$. The ith service provider S_i is defined as a finite set of offered services such that:

$$S_i = \{s_{ij} \mid j \leq J; j, J \in \mathbb{N}\}$$

The set of all services is $S = \bigcup_i S_i$ and the service chain is defined as a sequence:

$$C = (c_n \mid n \in \mathbb{N}) \tag{1.1}$$

where the terms are given by a generating function

$$f : \mathbb{N} \rightarrow S \tag{1.2}$$

Depending on the generating function f, a service chain may contain the same service s_{ij} at different places in the sequence. The same service executed at different times is denoted as s_{ij}^k where $k = 0, 1, \cdots, m$.

The application is defined as $A = \text{EX}(C)$ or a sequential[5] execution of terms from C, where c_m must finish executing before c_{m+1} starts executing.

[5]note that requiring a sequential execution is only introduced here in order to facilitate the derivation of an analytical expression and is not a limitation on service-based applications

1.3.2 Probability of Service Failure

Let the probability of obtaining a certain level of performance from service s_{ij} be represented by a random variable R_{ij} defined over the sample space Σ_{ij}. To make the analysis tractable, we require that the reliability of any service in the service chain C is independent of the reliability of other services, or for any event A_{ij}^k such that $R_{ij}^k(x) = a_{ij}^k$ where $x \in \Sigma_{ij}$ and $a_{ij}^k \in R_{ij}^k(\Sigma_{ij})$, then:

$$P(\bigcap_{l \in L} A_l) = \prod_{l \in L} P(A_l) \tag{1.3}$$

for any subset L of $\{1, \cdots, d\}$, where the number of terms in C is $d = |\{f_n \mid n \leq N\}|$.

The user can set a value of $0 < r_{ij}^{min} \leq 1$ which is the minimum acceptable reliability of a service s_{ij} as per assumption 2. Let the probability density function of the random variable R_{ij} be represented by some function g_{ij} defined over $[0, 1]$, then the event of the service failing, $R_{ij} \leq r_{ij}^{min}$, occurs with probability

$$q_{ij} = P(R_{ij} \leq r_{ij}^{min}) = \int_0^{r_{ij}^{min}} g_{ij}\, dx \tag{1.4}$$

Unfortunately, the function g_{ij} (and possibly its parameters) is not known and has to be either assumed or estimated.

Assuming a particular density implies some prior knowledge, perhaps of the behavior of the service provider. In the absence of such knowledge however, any function would be equally valid. Estimating it, on the other hand, presumes that some reliability data exists and can be used for that purpose. If the user repeatedly interacts with a service provider whose identity can be ascertained, then some machine learning algorithms can be implemented for that purpose. One direct consequence is that any performance comparison of the proposed method to improve robustness to other methods is highly sensitive to the selection of the density function.

1.3.3 Analytical Expression for Application Robustness

The probability of the event that the application is robust is a function of the number of times each service is used and their respective failure probability q_{ij}. The former is obtained by counting how many times the event $R_{ij} \leq r_{ij}^{min}$ occurs during T_A since a service s_{ij} may be used up to l_{ij} times. This is expressed by the binomial distribution $f(x) = \binom{m_{ij}}{x} p^x q^{m_{ij}-x}$ where $x = 1, 2, \cdots, m_{ij}$ and $m_{ij} \leq l_{ij}$. By assumption 3, the

Parameters	Remarks
i, j, m_{ij}	• Once the generating function determines the service chain, quantities represented by the parameters cannot be modified • The generating function itself might be defined in order to optimize some objective related to those parameters
q_{ij}	• Quantity represented by this parameter can be modified at anytime, and whose reduction is very desirable • This parameter is an estimate, and the robustness of the application will depend in large part on the accuracy of the estimating method

Table 1.2: Parameters controlling the robustness of the application

application will only work if $f(m_{ij}) = \binom{m_{ij}}{m_{ij}} p_{ij}^{m_{ij}} q_{ij}^{0}$ where $p_{ij} = 1 - q_{ij}$, thus $f(m_{ij}) = (1 - q_{ij})^{m_{ij}}$. Extending to all i, j, we obtain the probability of the application not failing,

$$P(\bigcap_{i,j} R_{ij} \leq r_{ij}^{min}) = \prod_{i}^{I} \prod_{j}^{J} (1 - q_{ij})^{m_{ij}} \tag{1.5}$$

It is readily observable that this probability will decrease with increasing m_{ij}, q_{ij}, i, and j. This suggests that to improve the probability of the application working properly, we must find ways to decrease the above parameters.

1.4 Problem Formulation

In order to formulate some strategies for decreasing the m_{ij}, q_{ij}, i, and j parameters, they are first separated into two groups. Once the generating function in Equation 1.2 establishes the service chain C, then i, j, m_{ij} are known quantities and the reduction of any of these will negatively impact the application from users' perspective. It is possible however, to envisage the generating function f to be optimized with respect to those parameters. On the other hand, q_{ij} is an estimated quantity whose reduction is highly desirable. This is summarized in Table 1.2.

1.4.1 Minimizing q_{ij}

Obviously, $P(R_{ij} \leq r_{ij}^{min})$ can be minimized by reducing r_{ij}^{min}, implying that users would have to lower their expectations. This is the trivial solution to problem of increasing the robustness of the application.

The probability density function of $g_{ij}(x)$ is characterized by some mean and standard deviation. If we could exert some influence through an incentive mechanism on μ and σ, then we could minimize the probability of the event $R_{ij} \leq r_{ij}^{min}$. This is the approach taken in federated networks, grid networks, peer-to-peer systems and generalized by the concept of incentive mechanisms such as reputation networks, for example.

A brief review of reputation networks as well as other possible methods dealing with providers' rationality are presented in section 2.1. A more in depth analysis is presented in Chapter 4 where a qualitative framework is introduced for comparing incentive mechanisms.

1.4.2 Minimizing i, j, m_{ij}

Taken together i, j represent the total number of services, and it makes sense to discuss them jointly. Thus there are two options, reducing the number of time a service is used, and reducing the number of services. As noted previously, either of those solution might materially affect the execution of the application, unless this is an explicit constraint on the generating function in Equation 1.2.

Automatic service composition techniques from Web Services and Service Oriented Architectures are an active research area potentially applicable to this problem as well. The generating function given by Equation 1.2, could be devised so as to produce a sequence as in Equation 1.1 subject to some relevant constraints. Service composition is presented in Section 2.2.

1.4.3 Proposed Approach

If the generating function cannot be modified, then the remaining possibility is to express all instances of at least one term s_{ij} of C as a function of other terms in such a way that the functionality of the application is not changed and the overall probability of failure decreases. The rationale for the proposed approach is based on the observation that services are supposed to be re-usable and independent of user's context, and thus there may be some redundancy between services which may be leveraged.

All the services $s_{ij} \in S$ that the application relies upon is given by $S_C = \{s_{ij} \mid f(n) = s_{ij}, 1 \leq n \leq N\}$. Let the target service $s_t \in S_C$ be the service to be replaced by a candidate service $s_c \in S_C \setminus \{s_t\}$. A solution will be a function h where $s_p = h(\{s_c\})$ such that $s_t \equiv s_p$ and the probability of failure of service s_p is less than the probability of failure of service s_t.

The desired outcome therefore is that we may be able to replace parts of the service chain of Equation 1.1 by a service based on a service already used such that the overall robustness increases.

1.4.4 Research Questions

The problem of finding the function h (transformation of one service into another) can be broken into the following issues:

R1 - Given all the services that the application depends on, how can they be classified into candidate (reliable) and target (unreliable) services? A framework which could be used to make that classification is discussed in chapter 4.

R2 - What is a suitable formal expression for services? Expressing services as finite-state machines allows the application of efficient algorithms and is discussed in Section 5.2.

R3 - In order to express a target service in terms of a candidate service, we try to re-use as much functionality from the candidate service as possible. How can this redundancy be described and found? This is discussed in section 5.3.

R4 - Using redundancy between services, how can we express a service in term of another service? This is discussed in section 5.4.

1.5 Thesis Contributions

The four contributions of the thesis, as shown by the publications in peer-reviewed conference proceedings and journals listed in section 1.7, are as follows.

C1 - Design of an qualitative framework to assess the impact of incentive mechanisms on the quality of received services, allowing classification of those services.

Without any incentive mechanism, and with basic assumptions, it is easy to show that if service providers behave rationally, their services quickly collapse. To avoid this outcome, many incentive mechanisms have been proposed. Each service necessary for an application, however, may be sensitive to a different incentive mechanism (for example, by belonging to different virtual organizations). A qualitative incentive reference framework is proposed to compare the impact of various incentive mechanism implementations on the received service.

C2 - Formalization of the concept of redundancy based on the finite-state machine notation, and an algorithm to obtain this redundancy from any pair of services.

The premise of services in the context of Service-Oriented Architectures is that they may be used in contexts not foreseen by the designers or creators of those services. It is likely therefore that some of the functionality will be duplicated between some services, or that some part of a service will be redundant with respect to another service.

C3 - An algorithm to express one service in terms of another using redundancy between the two services, based on the finite-state machine notation.

The algorithm works by decomposing the target service into three submachines such that the entire functionality of the original target machine can be recreated through application of the composition operation to the submachines. One of the submachines is a modified candidate machine while the remaining submachines represent the locally implemented functionality and control.

C4 - Design of service-based collaborative authoring application.

The basic requirements of the application are highlighted and fulfilled by utilizing certain services accessed through standard HTTP, Jabber and JXTA set of protocols, and off-the-shelf techniques. By measuring the performance of the application in a heterogeneous environment and by providing details of an alternate service fulfilling the application's requirement, it is shown that service-based collaborative applications can be quickly designed and deployed.

1.6 Thesis Organization

This thesis is composed of the following seven chapters; Chapter 1 provides the INTRODUCTION, comprised of the background material, motivation, problem and model description, problem formulation and contributions of this thesis;

Chapter 2 presents the state of the art in dealing with the effects of rational behavior, which comprise trust and reputation networks, and methods based on game theory and mechanism design. These are presented and discussed in a broader survey of available strategies, tools and available techniques to analyze and mitigate effects of rational behavior. As mentioned in section 1.4, possible ways to improve robustness include optimizing service composition methods, hence, the STATE-OF-THE-ART AND RELATED WORK chapter also includes discussions on service composition. The proposed solution is illustrated in the context of a collaborative application, and thus a brief survey of issues in collaborative environments is presented as well.

Chapter 3, DESIGN OF A SERVICE-BASED COLLABORATIVE APPLICATION, presents a working implementation of a collaborative application assembled from services. The key concept is that the application may be built from different services, depending on their availability and reliability, while preserving its functional requirements.

In the context of this thesis, the probability of failure of a service might be influenced by implementation of incentive mechanisms. Chapter 4 describes a qualitative INCENTIVES REFERENCE FRAMEWORK, which presents elements common to all incentive mechanisms, and choices available for their design. This framework provides a systematic way of thinking about the impact of incentive mechanisms on the quality of received services, and thus can be used in the classification of services.

The exact description of services, the concept of redundancy of one service with respect to another, and the algorithms used to express one service in terms of other services are presented in Chapter 5, LEVERAGING INTER-SERVICE REDUNDANCY.

Chapter 6, METHOD EVALUATION, presents an example of the usage of the method presented in this thesis. Based on the application of chapter 3, all required services are expressed as finite-state machines, and assuming that one service is classified as a target machine, it is expressed in terms of candidate services. Time and state complexity analysis of the method is also presented.

Chapter 7, CONCLUSIONS, summarizes this thesis and its findings, open questions and outlines future work, both from theoretical and practical point of view.

Appendix A provides the background to the FINITE-STATE MACHINES formalism. Finite-state machines are used to describe services, and certain operations are performed on services thus represented in order to express one service in terms of another.

1.7 Publications

In the course of writing this thesis, various aspects of its theme were published in:

1. Andrew Roczniak, Pierre Lévy and Abdulmotaleb El Saddik, "Improving Robustness of Service-based Applications by Leveraging Inter-Service Redundancy". (Submitted)

2. Andrew Roczniak and Abdulmotaleb El Saddik, "INCA: Qualitative Reference Framework for Incentive Mechanisms in P2P Networks". In International Journal of Computer Applications in Technology, Vol. 29, No. 1, pp.71-80, 2007

3. Andrew Roczniak and Abdulmotaleb El Saddik, "Mobile P2P Data Retrieval And Caching". In Encyclopedia of Wireless and Mobile Communication, Edited by Borko Furht, CRC Press, 2007 (to be published)

4. Andrew Roczniak, Alexandre Miège and Abdulmotaleb El Saddik, "Security Considerations for SOA-based Multimedia Applications". In Proceedings of the IEEE International Symposium on Multimedia (ISM 2006), December 11-13, 2006, San Diego, California, USA.

5. Andrew Roczniak, Jamil Melhem, Pierre Lévy and Abdulmotaleb El Saddik, "Design of Distributed Collaborative Application through Service Aggregation". In Proceedings of the 10th IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2006), Oct. 2-6, 2006, Torremolinos, Malaga, Spain.

6. Andrew Roczniak, Salinah Janmohamed, Christian Roch, Abdulmotaleb El Saddik and Pierre Lévy, "SOA-based Collaborative Multimedia Authoring". In Proceedings of the 2nd Montréal Conference on e-Technologies (MCeTECH06), May 17-19, 2006, Montréal, Canada.

7. Andrew Roczniak and Abdulmotaleb El Saddik, "JADE: Jabber-based Authoring in Distributed Environments". In Proceedings of the 13th Annual ACM International Conference on Multimedia (MM 2005), November 06-12, 2005, Singapore.

8. Andrew Roczniak and Abdulmotaleb El Saddik, "Impact of Incentive Mechanisms on Quality of Experience". In Proceedings of the 13th Annual ACM International Conference on Multimedia (MM 2005), November 06-12, 2005, Singapore.

9. Andrew Roczniak, Abdulmotaleb El Saddik and Pierre Lévy, "Survey of Collaborative Environments". In Proceedings of the 17th Canadian Conference on Electrical and Computer Engineering (CCECE 2004), May 2004, Niagara Falls, Canada.

Chapter 2

State-of-the-Art and Related Work

Chapter 1 identifies rational behavior as having an impact on robustness of service-based applications. A prospective service consumer needs to reflect this impact in both its functional and non-functional requirements, unless steps are taken to provide some system-wide solution.

The state of the art in dealing with the effects of rational behavior comprise trust and reputation networks, and methods based on game theory and mechanism design. These are presented and discussed in a broader survey of available strategies, tools and available techniques to analyze and mitigate effects of rational behavior.

The proposed solution is related to automatic service composition, an active research topic. It presents a method of service description and composition, but the number of composed services and the success criteria differ from those in more traditional composition techniques, reflecting the different goal that is sought. An overview of service composition methods is presented.

Finally, the concept of service-based applications is applied to collaborative applications. An active field of research investigates interaction and collaboration between people as afforded by the growth and prevalence of computers and networks. This research, termed collaborative environments (CE), draws from many different disciplines including multimedia, distributed systems and networking. CE encompass a variety of applications, each with its own set of requirements. Many of those requirements address issues in scalability, efficient communication, quality of service, synchronization and security. The following presents a brief survey on state-of-the-art of the research in the field of collaborative environments as applicable to those issues.

2.1 State-of-the-Art: Handling Rational Behavior

Service-based applications can broadly be characterized as dynamic networks of globally distributed, heterogeneous and autonomous entities that interact and cooperate through sharing resources and goals. Shared resources can be fundamental building blocks of Internet computing such as storage space and bandwidth [146], or in context of Service-Oriented Architectures (SOA), more abstract concepts such as services [111]. Shared goals, such as handling unpredictable computing storage space needs or access to applications, put requirements and constraints on systems supporting those virtual organizations. Many implementations however, assume that the goals are perfectly aligned for each participant, or in other words, that the understanding of what constitutes a "goal" for one entity is exactly the same for all other entities and that all have the same preference as to the outcome of the cooperation. Misalignment of participants' goals could result in unpredictable results, or at its worst, unacceptable situations for participants.

As described in Chapter 1, this misalignment of goals is induced by participants' "autonomy" (which allows us to model the behavior as rational) and has an impact on the robustness of service-based applications.

Autonomy does not imply that service providers will behave rationally; however it certainly allows them to be self-interested, in the sense that they will choose a course of action that leads to their preferred outcome. If, for example, a service provider wants to have its resources used but at the lowest cost to itself, it can consistently advertise its service with the highest performance characteristics (assuming users seek highest performance) while always offering an equivalent service that minimizes its own cost (*bait-and-switch*). A participant in a service-based application does not have direct control over this behavior, but may have some influence through some kind of incentive mechanism.

Robustness is often an intuitive predicate. Airplanes are thought of as robust (in the sense of not falling from the sky), otherwise there would be no passengers on board. Similarly, software is expected to be very robust when put in place in a nuclear power generating station, and less so when used in a family photo album. This expectation is influenced by the consequence and likelihood of failure and in practice this translates into corresponding amounts of resources, and hence money, dedicated to prevent that failure.

The motivation, thus, for this brief survey stems from our research into robustness of applications that depend on services provided by autonomous entities. Following the

realization that the behavior of participants plays an important role, we endeavor to map out possible strategies and techniques available to handle participants' rational behavior in service-based applications.

In the next section, we review strategies available to service-based applications' designers, wishing to take into account participants' behavior. If the selected strategy calls for leveraging participants' (rational) behavior, then the techniques presented in Section 2.1.2 can be deployed. Those techniques broadly fall into two classes, the genuine and artificial incentives; formal tools applicable to incentives in general and an example of artificial incentives, trust and reputation networks, are discussed in Sections 2.1.3 and 2.1.4 respectively. Finally, we present our conclusions in Section 2.1.5.

2.1.1 Strategies

As outlined in the introduction, the pursuit of self-interest by providers may impact the benefits derived from participating in a service-based application. If the ultimate goal is to be able to balance the application's desired robustness with an acceptable cost, then as a first step, an application specific analysis should be made of the impact of rationality on the benefits to participants. A second step would be to define the trade-off between a solution mitigating this impact and the cost of any implementation. In our research for example, we wish to achieve a certain level of application's robustness at an acceptable computational cost to participants, which naturally led us to consider the possible responses available to deal with rational behavior. The following strategies are based on the suggestions outlined in [134] and are summarized in Figure 2.1.

Ignore it

This do nothing alternative may be applicable where the cost of handling rationality outweighs potential benefits. Napster and Gnutella, the original file sharing applications, are practical examples of systems ignoring rational behavior while providing tangible benefits to users. Despite their documented shortcomings with respect to this behavior [7, 124], they were instead laid low by an extraneous force and technical progress respectively.

Eliminate it

This, in essence, entails encapsulating the responsibility for negotiating and enforcing that services are reliable to an outside system, through an out-of-band mechanism or partial centralization of some aspects [105]. The former can apply to a group of friends

or institutions entering into monitored service-level agreements (SLA [79, 27]), and is illustrated by the bounded-price mechanism proposed in [16]. This mechanism is deployed in environments with self-interested participants, and is based on contracts negotiated off-line with deviations assumed to be punished by monetary penalties. The second approach may rely on an omniscient and trusted third-party to identify participants and enforce contracts between them as proposed in [58]. This solution ties received services to a promise of participating in the cost of providing other services. The promise is codified in a contract and enforced by an entity that attempts to maximize the collective benefit received by all participants.

Hide it

The possibility of a service provider modifying its behavior is greatly reduced if trusted software and/or hardware is employed within the virtual organization. For example, [93] addresses the issue of behavior conformity - or the expectation that participants should behave in line with the organization's rules - in grid computing. Noting that conforming behavior can help increase overall security, authors propose using trusted computing technology from Trusted Computing Group (TCG) [143].

Avoid it

Identify and ignore participants deemed to pursue a goal markedly different from that of the rest of the community. As observed in [134], since failure handling techniques in traditional distributed systems differentiate between correct and faulty behavior, it could provide a basis for a mechanism to avoid rational behavior. Another approach, the focus of this thesis, is to allow an application to reuse some services in order to bypass a service deemed unreliable.

Use it

Specifically incorporate and leverage the behavior of service providers into the design of the application. The family of techniques that fall under this strategy, differentiate between the types of service providers' behavior. These can be classified [134] as *obedient* if they always follow the mandated protocol or specification irrespective of other considerations; *faulty* if they stop working or act arbitrarily; *rational* if their behavior is driven by attempts to optimize some function based on its knowledge and understanding of the context; and finally, *irrational* if they attempt to optimize some function that is

not explicitly modeled in the description of the system. A brief overview of available techniques is presented in the next section.

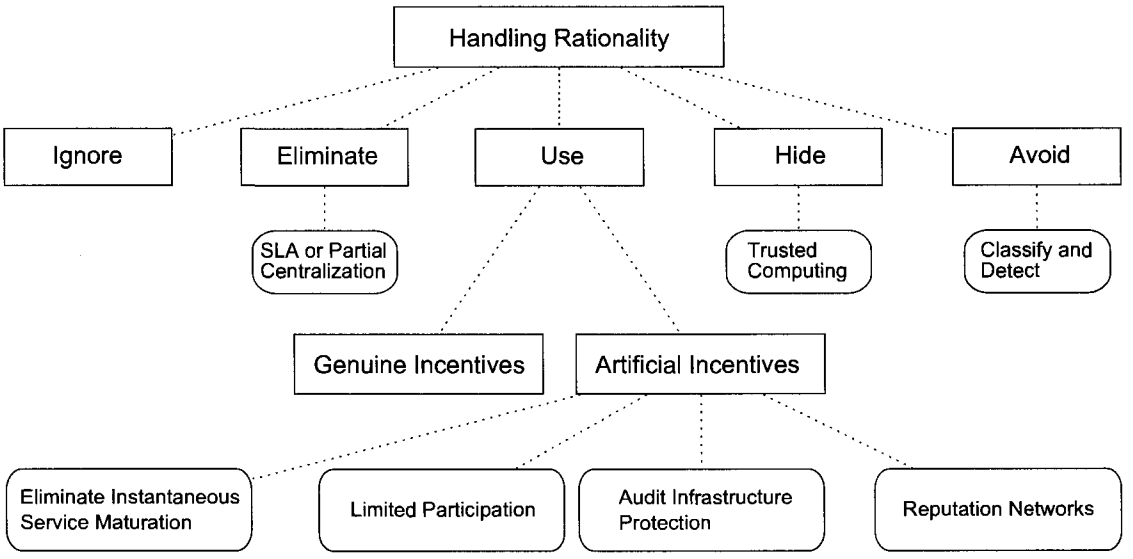

Figure 2.1: Handling rationality: possible strategies and techniques

2.1.2 Leveraging Providers' Behavior

Rational and irrational service providers are said to follow some strategy. The goal, therefore, of leveraging providers' behavior is to "build a mechanism that enables strategizing nodes to act rationally, and incentivize rational nodes to behave well" [134]. Formal tools to analyze and synthesize incentive mechanisms are briefly presented in Section 2.1.3.

With respect to a behavior that seeks to maximize the usage of system's shared resources (bandwidth in file sharing, for example), [105] provides a taxonomy of "rational attacks", and provides a finer-grained analysis of potential solutions by distinguishing between genuine incentives which are "characterized by directly incentivizing cooperation" and artificial incentives which "incentivize evidence of cooperation" [105].

Reciprocal mechanisms, where the interaction is repetitive and participants do not exactly know when it will end, are an example of genuine incentives. Generally, situations where parties to a contract exchange their obligations in small increments, reduce the possibility of participants not keeping their side of the bargain. In the context of the download of a file, an example of a working reciprocal mechanism is the BitTorrent [13]

protocol, even though available implementations of this protocol may not necessarily be *faithful* to its desired specification [133]. Other examples include systems based on tokens [66] or micro-payments [60]. Artificial incentives are arguably weaker than genuine incentives but correspondingly easier to implement and the following four methods are highlighted.

Eliminate instantaneous service maturation

If a service matures instantaneously, i.e., is consumed before any reciprocity is received, then the consumer of such a service can easily take advantage of the producer. Participants' behavior can be influenced by requiring them to leave a *security deposit* [147] or to *pay their dues* [114] before they can become service consumers.

Limit the number of entities

If nodes are capable of remembering the outcome of the interaction with other participants, and the interaction is repetitive - a situation which can be modeled as Iterated Prisoner's Dilemma (IPD [15]) - then [73] argues that assuming all nodes behave rationally, the best available strategy is retaliatory and encourages cooperation.

Protect the audit infrastructure

The evidence of cooperation must be protected if participants are to use it to make their decisions. This may be based for example on verification protocols [35] or securing shared history [52].

Deploy trust and reputation networks

Whenever a consumer needs to make a selection between comparable services, the decision often hinges on the belief in the accuracy and truthfulness of the claimed quantitative (or qualitative) criteria. This belief can be strengthened by repetitive satisfactory interaction with the provider or based on a record of interaction evaluation made by other entities. In settings with large population of services or entities, it may be difficult or impossible to form an opinion about each member of the population, and thus using others' opinions to guide one's decision quickly becomes an appealing solution. This extensive topic is briefly presented in Section 2.1.4.

2.1.3 Analysis and Synthesis Tools

Interaction between rational participants and impact of deployed incentive schemes are usually analyzed using game theory. On the other hand, synthesis of incentives is usually accomplished through *mechanism design* (MD). In the following, we briefly present salient points of these two tools using an illustrative example of children dividing chocolate bars amongst themselves.

Game Theory

A "game" is played by "players" using "strategies" that lead to "outcomes" which give each player some "payoff". Each player has a preference for some outcome (higher payoff) which may be given as a utility function u which maps outcomes to payoffs, $u : \mathbb{O} \rightarrow \mathbb{P}$. Players are rational in the sense that they will strive to maximize the utility function in every situation, i.e., obtain the highest payoff. Players will attempt to achieve that goal by playing a strategy, or taking actions, in response to every possible strategy other players might use. Outcomes are of course the result of this strategic interaction, and might not necessarily be what the players intended or desired.

Incentive mechanisms act by changing the payoff in certain outcomes, which hopefully changes which strategies a player will choose. In this light, some taxes or income deductions legislated by governments might be viewed as incentives to spend, save or invest.

Given the "rules" of the game, which may be diverse, game theory allows to analyze or predict results of the interaction. For example specifying how many children and chocolate bars there are, if the interaction recurs regularly and if it is always the same children that interact, and assuming that the utility increases with increasing amount of the chocolate bar, one possible result is that rationality will drive a child to hoard all the chocolate.

A set of strategies that lead to the highest payoff for each player may be viewed as a solution to that particular game, or an "equilibrium" upon which the game converges. A Nash equilibrium for example, is the set of strategies where no player can improve its payoff by changing its strategy given other players' strategies. A dominant-strategy equilibrium occurs when all participants have one strategy that is superior to all other available strategies, regardless of what other participants do.

Finite zero-sum games where perfect information - full knowledge of available strategies, payoffs and player preferences to outcomes - are relatively easy to analyze. However,

in most non-zero-sum games there are multiple Nash equilibria and not all information may be known by all players which increases the complexity of the analysis. For these and other reasons, some "are happy to recognize that game theory isn't useful for every decision problem, or even every strategic decision problem, that comes along" [122].

Mechanism Design

Mechanism design deals with the design of the rules of the game such that when rational participants with private information (which influences their choice of strategies) interact using those rules, their best choice of strategies lead to an outcome intended by the designer. For instance, if the goal is to have a chocolate bar divided equally between two children, then it can always be achieved when the rules specify that one child cuts the bar and the other subsequently chooses which piece to take[1].

More formally, a mechanism is defined as a tuple of all strategies Σ available to all participants n and an outcome function $f : \Sigma_1 \times \Sigma_2 \times \ldots \times \Sigma_n \rightarrow \mathbb{O}$. The mechanism will be designed for a particular outcome $o \in \mathbb{O}$ if $f(s^*) = o$ where $s^* = (s_1, \ldots, s_n)$ is an equilibrium solution to the game.

Outcomes can of course be varied, from maximizing sale price in an auction to efficiently allocating load-balancing or routing resources for example. What can be said about this outcome? Of particular interest to designers wishing to optimize utility gained by a group is Pareto efficiency. This predicate indicates if a player can change its strategy for a higher payoff without another player receiving lower payoff, or, in other words, that an alternative outcome exists that is preferred by at least one player while others are indifferent. Mechanisms themselves will have certain (desirable) properties, and for example, mechanisms are Pareto optimal if they implement Pareto efficient outcomes and strategy-proof if players' dominant strategy leads them to the desired outcome.

The situation where strategies followed by players do not naturally lead them to the desired outcome, are handled by a transfer function consisting of some kind of "subsidies" and "taxes" (incentive-compatible mechanism) which make it worthwhile for the player to choose strategies that lead to desired outcomes.

Two main issues underpin any successful implementation of a mechanism. The first, all players must at least obtain as much utility from participating as from not participating in a mechanism (all rational participants would abstain otherwise), and secondly, from a computational perspective, both the output and transfer function must be computed

[1]Those actions can be seen as an example of *commitment* in game theory

in polynomial time. Algorithmic mechanism design (AMD) and distributed algorithmic mechanism design (DAMD) treat the cases of function computation in a centralized and distributed manner respectively [51].

2.1.4 Trust and Reputation Networks

If a large number of participants make an evaluation, and this evaluation is a basis for future decisions, then we have solid foundations for designing and deploying artificial incentives. This approach however presents numerous challenges in practice, and for illustration purposes we highlight three of these [72, 139, 117]. First, the record must be found in a possible distributed environment, second, the precise evaluation methodology must be understood (collaborative filtering or recommender systems are possible otherwise [141]), and finally, a decision must be made as to how believable the evaluator is. Those challenges are usually addressed simultaneously, as illustrated by the following three examples.

Locating trustworthy services is discussed in [156]. A node starts by locally evaluating (or ranking) the quality of the services or referrals of other nodes it knows about which is then used as a basis for neighbor selection policies. Since locating a specific service (or a service with specific characteristics) can be modeled by a search on a graph, the authors discuss the suitability and performance of various graph topologies, referral and neighbor selection policies.

In the context of P2P file sharing networks, [75] presents an algorithm to aggregate local trust values (number of satisfactory transaction less number of unsatisfactory transactions) into a global reputation value. This is achieved through weighing other participants' reported local trust values by the local trust a node has in that participant. Authors report that this value is obtained relatively fast, which reduces the amount of required overhead traffic.

Noting that trust and reputation are context dependent, multi-faceted and dynamic, [150] proposes a Bayesian model for representing the trust in services or recommendations of a specific participant. Authors note that this solution is applicable in cases where there is repetitive interaction between participants, either in small networks or large networks that exhibit small-world characteristics.

The above methods are furthermore sensitive to security issues, from protecting the reputation data [43] and anonymity to identity [45, 55, 89].

2.1.5 Discussion

With the current trend of using services or resources that do not directly fall under the control of one authority, an entity may need to make adjustments to its requirements to account for rational behavior. Designers of service-based applications are perfectly placed to decide how important this issue is to their applications, and what is the best solution given their goals.

To help in these decisions, we have presented a brief survey of possible strategies, from ignoring rational behavior to leveraging it, techniques such as trust and reputation networks to mitigate its impact, and formal tools of game theory and mechanism design to respectively assess the outcome of an interaction, and define the rules of an interaction in order to attain an outcome desired by the applications' designers.

2.2 Service Composition

Composing services in order to create other services presumes that the available services provide some kind of description of themselves. There are numerous ways to achieve that, from specifying the behavior of a service to describing the interface (input and output) along with, potentially, the context, preconditions and postconditions.

Automatic service composition methods additionally assume that the service description is represented not only in a machine-readable way, but also in a formal form amenable to algorithmic procedures, either directly or through some kind of (algorithmic) translation.

Obviously, the best solution to the automatic composition problem would allow, at the same time, rich expressive semantics (to express functional, non-functional requirements and constraints), a formal expression of correctness and completeness, and of course, scalability of the composition method as the number of available services grows.

As automatic service composition area is currently of great interest in both academic and industry circles, numerous articles present in more detail the issues and techniques in service composition. Interested readers are referred to e.g. [12, 140] for a more in-depth analysis. Based on [96], a a brief summary of methods is presented in Section 2.2.1, and two specific examples of methods verifying the existence of a composition are discussed in Sections 2.2.2 and 2.2.3.

2.2.1 Automatic Service Composition

Analyzing automatic service composition from the point of view of four key issues, connectivity (a service may be found and interacted with), non-functional quality of service (QoS) properties, correctness (verification that the composition matches the requirements) and scalability, the authors review the following approaches for service composition:

- Business Process Execution Language (BPEL), which enables "users to describe business process activities as Web services and define how they can be connected to accomplish specific tasks" [106]
- OWL-S which aims to "facilitate the automation of Web service tasks including automated Web service discovery, execution, inter-operation, composition and execution monitoring" [33]
- Methods based on algebraic process composition [65, 53]
- Methods based on Petri nets [64, 145]
- Methods based on Artificial Intelligence (AI) Planning, where the service composition problem can be expressed as a planning problem [116]. This is formally represented as a five-tuple (S, S_0, G, A, Γ) where S is the set of all possible states, $S_0 \subset S$ is the initial state, $G \subset S$ is the goal state of the planning system, A is the set of available state transition actions which are related to the states by $\Gamma \subseteq S \times A \times S$. With respect to the service composition problem, S_0 and G are specified by the user and A is a set of available services

2.2.2 Example of a Method based on Theorem Proving

An example of theorem proving approach is the method presented in [115] which uses web services standard languages for *external* representation of atomic or composite web services, and translates them into an *internal* representation consisting of Linear Logic (LL) axioms and a statement of provability corresponding to the requirements on the composite service. Theorem proving is then applied to determine if the requested service can be composed. If a proof of the theorem exists, then a process model is derived from the proof, represented by a process calculus based on π-calculus. Finally, the construction of a flow model is obtained by translating the process calculus to OWL-S or BPEL.

The internal representation allows to express functional as well as non-functional attributes, and guarantees the correctness and completeness of service composition process. The type of logic suggested by the authors (MILL or multiplicative intuitionistic additive fragment of LL) is PSPACE-complete in certain instances. According to authors, the performance is comparable to other AI planning languages.

2.2.3 Example of a Method based on Finite State Machines

An example of automatic composition of services specified as finite-state machines is presented in [20]. A client expresses its requirements (referred to as target service) through an *external schema* which is modeled as a deterministic finite-state automaton $M_{ext} = (\Sigma, Q, q_0, \delta_{ext}, F)$ where Σ represents the possible actions; state transitions are given by $\delta_{ext} : Q \times \Sigma \rightarrow Q$; termination of a service is denoted by F where $F \subseteq Q$.

The specification meeting the client requirements is expressed as an *internal schema*, which is basically the external schema enhanced with the information on which service instances in the community execute each given action. Given a community $C = \{c_1 \cdots c_n\}$, the internal schema is a finite-state transducer $M_{int} = (\Sigma, \Gamma, Q, q_0, \delta_{int}, \omega_{int}, F)$ where Γ is the output alphabet $\{1 \cdots n\}$ corresponding to a specific $c_{i \in \Gamma} \in C$ and $\omega_{int} = Q \times \Sigma \rightarrow \mathcal{P}(\Gamma)$ returning a set of identifiers of services that execute a specific action.

A composition of an external schema E_{ext} with respect to the community C is defined as an internal schema E_{int} which satisfies the following conditions: $\omega_{int}(\sigma, q) \neq \emptyset$ (otherwise some actions cannot be performed by any member of the community); E_{int} satisfies some conformity to E_{ext} criterion; E_{int} satisfies some coherence with C criterion.

A Deterministic Propositional Dynamic Logic (DPDL) formula, polynomially related to the size of the automata in C, is shown to be satisfiable if and only if there exists a suitable composition E_{int}. Based on the decidability properties of the satisfiability of a DPDL formula, the decidability of the existence of a composition can therefore be obtained in EXPTIME.

2.2.4 Discussion

Service composition addresses the issue of creating a service from other services. As such, it can solve different problems, from integration, reuse and simplification of complex systems. There are various ways to describe a service and approach the problem of finding an acceptable composition, from manual to automatic. As noted in [96], most commercial methods emphasize the descriptive power of method to express functional and

non-functional requirements, while most academic methods concentrate on algorithmic composition with its attendant scalability problems. Most available academic methods also strive to be general, and thus, proving the existence or non-existence of a composition (a goal service obtained from component services) is paramount. In contrast, the proposed method is designed to compose services in order to minimize redundancy between services, and thus does not need to make any claims about the existence of a composition.

2.3 Introduction to Collaborative Applications

Collaborative environments can be classified according to three features, synchrony, locality and scale [126]. Synchrony defines the support of real-time interactions. Email is an example of an asynchronous application while Voice-over-IP is an example of synchronous interaction. Scale defines how well the system handles growing numbers of users, and locality defines where the deployment occurs; dedicated networks or over the Internet.

Application domains for collaborative environments include medicine, science, education, art, and distributed simulations. Collaborative environments are the result of interaction of various disciplines: multimedia, networking and distributed systems, and share many interesting issues such as scalability, interactivity and security.

With increasing number of users, issues in scalability of the system and its efficiency in supporting communication between participants need to be addressed. Benefits and trade-offs of multicasting, as well as congestion control and reliable multicast transport are discussed in section 2.3.1.

Whenever shared objects are modified, participants eventually need to be notified of that fact. Communication delays and lost messages among other issues affect the level of interactivity afforded by collaborative environments and is discussed in section 2.3.2.

Finally, collaborative environments can be used to share corporate or otherwise sensitive data, and hence some security level might be required, which is discussed in section 2.3.3.

2.3.1 Multicasting

Multicasting provides efficient communication within a group, specifically when concurrently sending the same data to multiple recipients. Efficiency is defined as better band-

width utilization and lower router processing load. Multicasting can be implemented on UDP or by developing new multicast oriented transport protocols. A classification of multicast protocols can be found in [107] and the following presents only salient points.

Reliable Multicast

Reliability in the context of this document means that all receivers eventually obtain all packets generated by the sender. In real-time video or audio applications, reliability is often traded for timely delivery. Classification of reliable multicast protocols can be found in [88]. The following presents a brief overview.

- Sender-initiated protocols. This class of protocols tracks the membership of the group and retransmits packets that were lost or corrupted. Retransmission occurs when the sender does not obtain an explicit acknowledgment within a certain period of time from the receiver. Without optimizations or other changes, this mechanism does not scale well due to the amount of processing required on the sender-side to manage all the acknowledgments.

- Receiver-initiated protocols. This class of protocols places the responsibility on the receivers to notify the sender of lost or corrupted packets. Receivers do this by sending a negative acknowledgment to the sender. Since variable delays can occur on the network, the sender has to keep copies of all previously sent packets.

- Hierarchical protocols. This class of protocols assumes that the receivers are organized in a hierarchy where the root of each hierarchy is responsible for retransmission of lost or corrupted packets. This class of protocols does not require the sender to know the entire group, and provides better end-to-end delays. It does however require some receivers to agree to act as root node.

- Ring protocols. This class of protocols requires a single token site responsible for sending acknowledgments to the sender and receiving negative acknowledgments from other receivers. This mechanism was originally developed to provide total ordering and guarantee atomicity.

- Multiple multicast groups. This class of protocols uses multiple multicast groups to send the original data and retransmissions. Depending on other requirements, the exact or approximate number of multicast groups can be calculated. These

protocols suffer from high overhead in processing join and leave operations of receivers.

- FEC-based protocols. This class of protocols, the sender transmits redundant data that allows receivers to reconstruct missing data. This mechanism lowers the impact of lost packets in real-time transmission.

Flow and Congestion Control

The impact of reliable multicast traffic on competing TCP traffic on the Internet in times of congestion is critical. Best-effort traffic responds to congestion on a link indicated by packet drops by reducing the load presented to the network. Reliable multicast (assuming it is not TCP-based) needs to implement flow and congestion control mechanisms in order to compete fairly with best-effort traffic. Classification of protocols can be found in [153]. The following presents a brief overview.

- Window based. Similar to TCP congestion control mechanism, this class of protocols uses a congestion window to regulate the amount of traffic presented to the network.

- Rate based. This class of protocols uses feedback mechanisms to indicate network congestion. Additive Increase, Multiplicative Decrease (AIMD) or model-based schemes can be used to control the amount of traffic presented to the network.

- Single/Multi Rate. In single-rate, the protocol sends data to all receivers at the same rate. In multi-rate, receivers join or leave multicast groups depending on congestion levels between sender and receiver. In video transmissions, joining more multicast groups can enhance the quality of the video, while for bulk transfer it can reduce transmission time. Congestion control is performed indirectly by group management and routing protocols of Network layer.

- End-to-End or Router-supported. This class of protocols differentiates itself based on additional information provided by network infrastructure.

Ordering

Order specifies the sequence of delivery of packets to an application. Ordering requirements can be causal or total. Causal ordering ensures that messages follow a specific

logical flow. An efficient method is the vector time-stamp as described in [125]. Total ordering requires that all receivers obtain all packets in the same order, and is necessary in some applications using distributed databases.

Application-Level Protocols

Real-Time Transport protocol (RTP) is currently used by the majority of tools for streaming audio and video over the Internet. RTP provides a flexible framework that is easily adapted to various applications' requirements. Interactive applications, where interactivity is defined as the ability to change the state of an application by supplying external events (such as user actions in a whiteboard application or networked games), follow a different media model than the one RTP was developed for. A new application-level protocol (RTP/I), specifically developed with interactive media in mind is presented in [95].

2.3.2 Consistency of Shared Data

Collaborative environments often use, create or access shared data, potentially resulting in global inconsistency if proper concurrency control mechanisms are not implemented. Propagating updates within a finite period of time, Collaborative Environments have to deal with two incompatible requirements [137]:

- High responsiveness. Perception of quality depends on the timeliness of response of the system to other users' actions.
- High concurrency. Concurrent actions of users do not result in a globally inconsistent state, even in the presence of conflicts.

Notification determines when, what, and how updates generated by one user are propagated and made available to other users [132]. If the information is replicated, all changes must eventually be propagated to every node [127]. Propagation mechanisms are generally separated from propagation policies. A propagation policy determines when updates should be propagated and depends on specific requirements of the application.

Conflicts will arise in replicated systems, and a mechanism to resolve them has to be considered. Conflict resolution can be automatic, manual or can be based on data or application semantics. Two approaches for dealing with data consistency in collaborative environments are available, the process group model and the transaction model. A third approach, based on *floor control* techniques is not discussed.

The process group model presents the distributed system as a collection of users that communicate by sending multicast messages. Techniques for ordering concurrent messages as well as ensuring atomicity of message delivery in the presence of delays and link failures, and changes to group membership need to be addressed. Some of the context has already been provided in the multicasting section of this document. Token based algorithms such as token Ring Protocol or Reliable Multicast Protocol are described in [36] and [152]. The token site orders the messages and broadcasts acknowledgments that are used by the destination sites to deliver messages in designated orders. Process group model offers different levels of consistency in terms of event ordering. Weaker orderings provide better performance, but at the same time, may increase the burden on the application designer to ensure correct executions.

The transaction model is characterized by operations being grouped into transactional units, with techniques available for ordering concurrent transactions and ensuring their atomicity. Transaction processing systems are widely deployed in concurrent and fault-tolerant access to persistent data. In groupware editing of a document, a sequence of changes made to a portion of the document by one user would need to be protected from other users' actions in order to reflect the desired changes correctly. If a transaction service is available within a collaborative environment, the transaction model offers simplicity in terms of application design. Some of the drawbacks to be considered are computational overhead and restrictive concurrency control [102].

As can be expected, neither traditional transaction processing, nor traditional distributed computing techniques, can satisfy all applications requirements. Some applications are better suited to transactions - if high concurrency control is needed - while group multicast better coordinates others - when real-time response is the primary design goal.

2.3.3 Security

Security issues in collaborative environments include authentication, authorization, data confidentiality and integrity and possibly non-repudiation. Additionally, flexibility and scalability characteristics of any solution have to be addressed. The following discussion presents some of the issues and existing solutions [9].

Authentication implies that both receiver and sender can establish each other's identity. In certain cases, only authentication of the sender might be required. Authentication can be ensured through public key infrastructure where issues of concern are

management, generation, distribution and destruction of private keys. Other mechanisms (Message Authentication Codes for example) are discussed in [34].

Authorization implies that a given user (previously authenticated or not) has the permission to perform certain operations on certain resources. To simplify security administration, a set of roles are created and users are assigned certain roles. Roles, and not users, are granted privileges within the system. A way of dealing with consistency checking of constraints is presented in [23], while issues in dynamic addition of users to a collaborative environment, is presented in [10].

Data confidentiality implies that only authorized users will have access to (possibly) encrypted data. Encrypting data and sharing the associated cryptographic key can ensure confidentiality. Within group communication context, a trusted group server distributes a group key to all members of the group. This group server periodically issues new keys, as well as when members join or leave the group. While negotiating new keys when a new member joins the group is straightforward, changing the key after a member leaves the group is computationally proportional to the group size and frequency of leave operations [154].

Data integrity implies that the received data has not been modified during transport. Integrity can be ensured on a per-packet, per-block or per-flow basis by including hash or digest information signed by a private key. Ensuring integrity of data flows (streaming), adds complexity since signing individual packets can be inefficient, while the potential of an infinite stream prevents validation at the end of transmission. In [57], a method for digitally signing streams is presented. This method relies on information present in previous packet to validate a current packet, and hence implies the need for reliable transport. Another technique does not assume the reliability of the underlying transport and is presented in [155].

2.3.4 Summary

This section presented a brief survey of research in the area of scalability, interactivity, resource management and security. Proper understanding of these issues is the basis for research and development in the area of Collaborative Environments.

Chapter 3

Design of a Service-based Collaborative Application

Developing a collaborative authoring application presents numerous challenges, one of them being that users' requirements may change faster than the development and distribution cycle of any application designed to meet their needs. Inspired by service-oriented architectures (SOA), the following investigates the possibility of empowering users to quickly assemble their own application from a set of standard services.

This chapter is structured as follows: section 3.1 introduces the concept of service-based applications in general and collaborative application in particular; sections 3.2 and 3.3 provide implementation details and experimental result for the collaboration and bulk transfer protocols respectively; section 3.4 presents security considerations for the resulting application; section 3.5 places the application prototype in the context of related research and discusses an alternative approach; finally, section 3.6 concludes.

3.1 Service-based Collaborative Applications

This section introduces the concept of building a collaborative application from services, presents an implementation of a prototype using specific technologies and its performance evaluation.

3.1.1 Conceptual View

At its basic, a collaborative authoring application needs to offer three main functionalities to its users: allow the sharing large amounts of data (e.g. share video clips), provide

group notification (to notify other users about one's actions) including participants' presence information (joins, leaves and availability), and ensure document consistency (to guarantee that all users agree on the state of shared objects). Other functionalities may be required, for example to allow users to continue their work off-line and to synchronize with others at a later time. The problem is thus to define services that will offer the application's required functionalities either directly, in combination with other services, or indirectly through some kind of adaptation on the application level.

Data Transfer. This functionality is necessary if one participant wishes to share its files with other participants, and the basic issue is how to transfer data from one user to a group of users. Many solutions are available, from centralized to distributed, and a service may implement any of these. A simple solution consists of a server that accepts uploads of files and allows download of those files on request. By using such a service, individual participants will not have to bear the cost of uploading potentially large amounts of data to a group of users.

Group Notification. If a collaborative effort of a group is synchronous (i.e. takes place at the same time), members of a group need to be aware of each other's actions. This is very similar to the data transfer functionality, except that the exchange is much more frequent and the size of transferred data is much smaller. A simple solution consists of a server that accepts an event notification from a user, and makes it available trough a push or pull mechanism to other members of the group.

In synchronous collaboration it is necessary to establish and maintain the membership of a group. At a minimum, the information about users leaving (including failures) or joining the group needs to be distributed to all current members. This concept can also be extended to notify the group about specific *interest* of a particular member.

Data Consistency. Whenever users can modify objects, the application must offer a functionality that ensures that eventually all agree on the state of the object. In this particular case, we have two options. The first is to design a service that either allows only one user at a time to make a change or a service that reconciles what each user thinks the state of a shared object is. The second, is to use the two previous services, group notification and presence information, to distribute the agreement on consistency over all the users.

The conceptual view of our aggregation framework is shown in Figure 3.1, where *Service State Support* provides a facility to keep track of the interaction with some services, and *Service Interaction Interface* enables the interaction with a given service. There may be numerous comparable services to choose from, depending on users' preferences

or requirements.

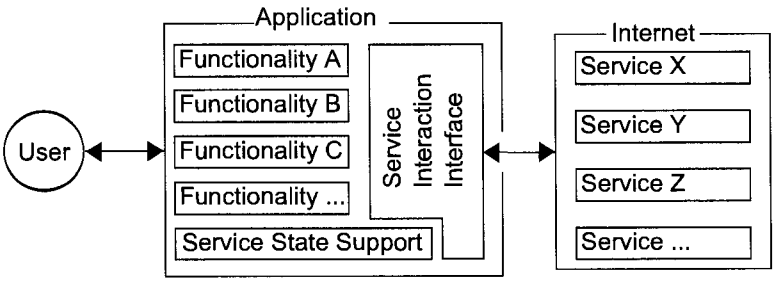

Figure 3.1: Conceptual view of services-based collaborative application

3.1.2 Design View

The problem is now reduced to selecting technologies that fulfill these three main requirements while being consistent with SOA philosophy.

A Jabber server provides various functionalities accessed through the Jabber set of protocols, and we are interested in three of them. The first, is an ability to deliver short XML-based messages to any other connected client. Second, Jabber servers may provide chat rooms supporting multiple participants where any message issued to the chat room will be forwarded to all participants that joined it. Finally, since each client needs to establish a connection with a Jabber server, this presence information may be made available (depending on security and privacy policies in effect) to other clients. We are not using certain functionalities such as chat room playback and potential server-side components as they are defined and controlled by the Jabber server administrator and may not be standardized.

We also use the JXTA set of peer-to-peer protocols to publish service advertisements at a Rendez-Vous (RV) peer, and to establish a direct connection between any two participants. Services are peer-based (RV acts as a broker), and we are mainly interested in the ability to exchange large amount of data between any two peers.

The group notification requirement can be met by Jabber's ability to send messages to individual participants as well as to the whole group, but we still need to address the bulk transfer and ownership management requirements. By design, Jabber servers do not handle transfers of large amount of data between any two peers in order to provide efficient and fair routing for all users, hence, we have implemented a HTTP-based transfer

using a Web server and a more efficient solution for larger files, a JXTA-based *bulk transfer protocol* that distributes the cost of uploading between all participants using Extended Content Management System (CMS).

The final requirement is met by leveraging the existing group notification service in conjunction with the presence service to implement a Jabber-based *locking protocol* to distribute ownership management over all participating users. This design view is illustrated in Figure 3.2.

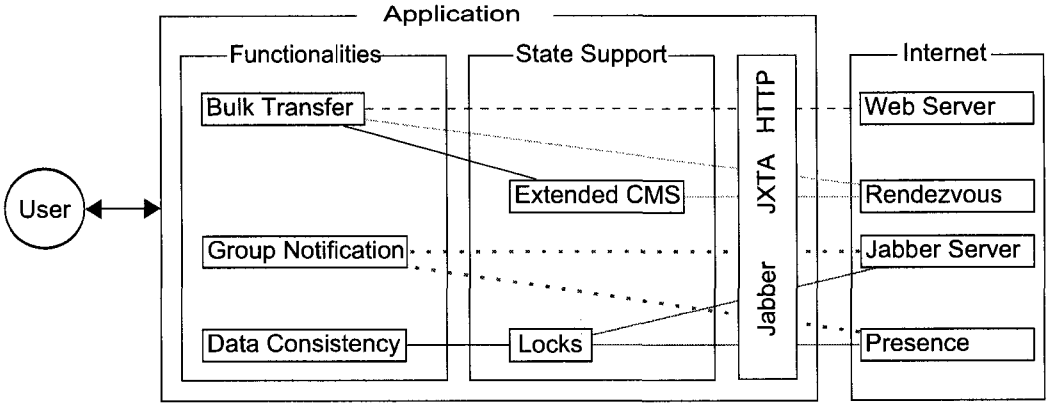

Figure 3.2: Design view of the multimedia authoring application

3.1.3 Deployment View

Services must be uniquely identified on the Internet, but there is no restrictions as to where exactly they must be deployed. It is perfectly possible to have all required services for a particular application co-located on one machine, or distributed across the users of the application themselves. From the point of view of the application therefore, it can follow a client-server or a peer-to-peer pattern.

3.1.4 Motivation for Protocols Selection

Traditionally, SOA implementation relies on the Web Services (WS) family of protocols to support interface definition and binding (WSDL), message exchange (SOAP), service discovery (UDDI) and on composition techniques as presented in [47].

The interface to the bulk transfer service for example, can easily be described in WSDL and interaction regarding the particulars (identification of users, group, session

and file names) can be based on SOAP. For the raw data transfer however, SOAP over HTTP will introduce unnecessary overhead. Group notification on the other hand, can leverage WS-Notification specifications which standardize event-based interactions, but introduce a fair amount of overhead.

We have elected to use the Jabber and JXTA set of protocols instead of the WS protocols due to their relative simplicity and their closer focus on the functionality we are developing and thus must explain which features of these protocols make our approach service-based.

Jabber is a set of streaming XML protocols that enable participants to exchange structured information in close to real time. Two of those protocols in particular are the extensible messaging and presence protocol (XMPP) defining messaging, presence, and request-response services between any two network endpoints, and instant messaging and presence protocol (XMPP-IM) defining instant messaging and presence functionality. These protocols are usually implemented over TCP/IP and are specified by the Internet Engineering Task Force (IETF).

JXTA is a set of protocols that specify basic functionality required by peer-to-peer type applications, and that inherently follow the *publish*, *find* and *bind* paradigm. Functions such as peer discovery, peer communication and their associated management are performed by publishing and exchanging XML advertisements and messages between peers. JXTA architecture provides a standard service discovery framework, allowing 3rd party services to be offered. An example of such a service is the JXTA Content Management System (CMS) which is used to manage and transfer shared files.

To verify that the usage of Jabber protocols is consistent with the general understanding of SOA, we are relying on the definition of a web service [90] and on compliance to the Reference Model for Service-Oriented Architecture as specified by OASIS [1].

The Jabber set of protocols are deigned with instant messaging applications in mind, allowing both person-to-person and computer-to-computer conversations. As such, the basic functionalities are very simple to understand and information about their invocation is often implicit (need for a username/password pair and the specific TCP port number on the server). Other functionalities and description about their invocation however (e.g. chat rooms), are discovered through a standard, XML-based and machine readable syntax. Interaction with these functionalities uses Jabber over TCP where all the allowed input and output messages are tightly defined.

The central concept of service-oriented architectures is that needs and capabilities are brought together. We make the basic assumptions that the server is willing to

engage in service interaction and that there is a working communication path between the producer and consumer. A consumer is aware of the existence of a server and its specific services by consulting a list of public servers and discovering the capability of each server by interacting with it. The effect of using services is very clearly defined and understood, although the implementation of each Jabber server may be very different (e.g implementations in C and Erlang). The specific functionality of chat rooms can be located and accessed (using IP and DNS) on a server different from from the one the client logged on to.

3.2 Document Consistency

Collaborative applications need to strike a balance between two incompatible requirements [137], high responsiveness and high concurrency. Perception of quality depends on the timeliness of response of the system to other users' actions, whereas concurrent users' actions cannot result in a globally inconsistent state, even in the presence of conflicts. Our group notification implementation focuses on the high concurrency requirement, and we assess the high responsiveness requirement from application usability perspective. Group notification determines when, what, and how updates generated by one user are propagated and made available to other users [132]. The implemented notification mechanism conforms to the process group model - the distributed system is treated as a collection of users that communicate by sending multicast messages. Ensuring message delivery in the presence of delays and link failures is provided by the Jabber protocol implementation while our implementation addresses the issues of concurrent message ordering and changes to group membership. A notification policy determines when updates should be propagated, and depends on specific requirements of the application. In our prototype, users' actions trigger an immediate update, if allowed by a locking mechanism. This locking mechanism is a type of floor-control technique [44], which should prevent conflicts from arising. In the rare case a conflict does arise from an attempt to obtain a lock, it is automatically resolved by our prototype.

The main collaborative functionalities in our application are provided by the Jabber Collaboration Interface (JCI) component. To avoid the inclusion of authoring semantics in JCI, a Document Object Model (DOM) is used to mediate between JCI and the authoring application. The JCI component has two main functions. First, it provides awareness of other participants' actions and contributions by representing interaction between users in a collaborative space. This space is application-specific and is concep-

tualized by some collaborative space metaphor. Second, it ascertains that a particular placement of an object will not generate conflicts as defined by the DOM. The document object model is an XML file specifying the relation of multimedia objects to the collaborative space metaphor. For instance, an image will occupy a certain area at certain position of a specific surface on a 3D object. For temporal objects, another variable would specify the duration of this relationship. DOM is the interface between the application and the JCI component by describing where and how conflicts may occur. For testing purposes a DOM that defines sixteen areas to which images are added is used, and as the end result of the collaboration, a collage of pictures is obtained. Finally, our application-specific code implements the collaborative space metaphor (single sheet of paper with predefined areas) and provides semantics for user actions such as multimedia object creation, modification and manipulation, which allows users to add, remove and order multimedia objects. Ordering can be spatial, temporal or both, as defined by the objects' characteristics and collaboration goals.

3.2.1 Collaboration Protocol

In order to maintain a consistent document during the authoring session, any modifications made in the collaborative space by a user must be coordinated with the actions of all other users. If a lock request is accepted by all the users, a modification may take place and becomes public after an unlock message. The collaboration protocol implemented in the JCI component conducts the locking and unlocking of spatial areas as defined in the DOM. The mutual exclusion thus achieved is a variant of the algorithm specified in [118]. The collaboration protocol uses time-stamps to resolve any conflicting lock requests. For each user, the JCI component maintains a list of locks that are currently in place for all users along with a list of pending requests that this particular user has made. Requests and replies are carried by group chat and individual messages respectively. There are five types of messages that the protocol defines and accepts. These simple XML-based message types are lock request, accept, deny, lock confirm, and unlock. All messages carry information about the issuer's identification, time-stamp and area identification as specified in the DOM.

Time Synchronization

When two requests conflict, the protocol accepts the message with the earliest time-stamp. In order for the time-stamp comparison to be relevant, it is necessary that users

agree on a standard time. In order to get all the nodes to agree on time, we use a light time synchronization protocol based on Cristian's algorithm [54]. One client node acts as time server, or leader. The leader sends out a series of values issued from the Java call *System.nanoTime()*, whenever a user joins the conference room, or every two minutes; other nodes do not request them. Every user in the room, including the leader, takes whichever time-stamp came through within the shortest delay out of the series, and uses it as a reference when a time value is required.

Leader Election

In order for the nodes to obtain a reference time, a leader has to be elected. If a user joining the conference room is the first participant, then this user is designated as the leader. If there are other participants in the room, the newly joined user will wait for a leader declaration message. Upon receiving this message, the latest version of the DOM is retrieved and the collaboration continues. In the case where the leader leaves the room, all the participants stop issuing requests, wait for any locks to be released, and elect a new leader. A new leader is determined by selecting the user who has the highest hash code of their unique user name. Since each user has access to a list of participants in the room, this calculation returns the same result for all users. Once the new leader has been selected the collaboration can continue once again.

Handling Group Changes

Group changes occur when a user joins or leaves the conference room and both events are handled similarly. Late join arises when a user joins a conference room that supports an on-going collaboration. This user then needs to obtain the latest DOM, which introduces two issues, obtaining the latest version of the DOM and ensuring that the DOM does not change while the new user is in the process of acquiring it. A file that meets both of those requirements is the authoritative version. Similarly, when a user leaves, its outstanding locks need to be deleted, and an authoritative version posted. Although all users have the same DOM and could therefore provide their copy, it makes conceptual sense to have the leader responsible for providing the authoritative version. When a user joins an empty conference room, it becomes the leader and retrieves and loads the DOM into memory. From then on, the leader is responsible for posting authoritative versions of the DOM. When a user joins a conference room that is not empty, it awaits a leader declaration message. Before the leader sends this message, it waits for all locks to be

unlocked, while all other users stop issuing new requests (replies and time-outs are still processed). When all locks are unlocked, the leader posts the DOM and issues the leader declaration message. The new user then retrieves the authoritative version of DOM from the designated web server. During the course of collaboration and in the absence of any group changes, processing an unlock message will update the version of the DOM that the user has in memory. Unlock messages hence contain all necessary and sufficient details of the modifications made to the multimedia document.

Deadlocks

In both the leader election and group changes procedures, clients might have to wait for locks to be unlocked, which presents a potential deadlock problem. If a client shuts down or its connection to the Jabber server is severed (due to a failure or by design), the server detects that the TCP connection to that client has been closed, and assumes the client has left. The server then sends a presence notification to the other participants, which allows clients to discard all pending and active requests owned by the disconnected client. If, however, a client *hangs* (e.g. malicious client) and stops responding while the TCP connection is still alive, requests emitted by other peers would continuously time out. To handle this case, other participants would need to agree which client failed and requires an addition to our protocol. A second case when a deadlock can occur is when a client connects to the server and receives a message saying it has joined the group, but is not yet in the group when requesting the list of participants. This is an artifact of the Jabber server implementation found out during the development of our prototype and happens very rarely. Although not implemented, this case could be handled by ensuring that the client cannot proceed if it is not present on the conference room roster.

3.2.2 Setup and Experiments

We have simulated[1] seven users collaborating over the Internet and the evaluation is based on the following four metrics. First, we measure the average time required to lock an area by subtracting the time at which a request was submitted from the time at which it was accepted by every collaborator, providing us with a lower bound for the delay in updating the multimedia document. Second, we measure the delay the user may experience while performing an addition to the multimedia document by taking the average total time to request a lock, obtain, it, upload a relevant object, and send out

[1] although 7 computers were used, their 'users' were scripts

the update. Third, we count the proportion of non-successful requests which require the user to reconsider the change they're trying to make and possibly redo it. Finally, we obtain those metrics when the size of the group varies from three to seven collaborators.

We have implemented a script that simulates users' behavior by including the following four actions: joining a chat room and downloading the latest version of the DOM, initiating changes to the document, updating the local copy when updates are received from other users and finally, saving a backup copy of the document whenever somebody leaves the conference room. Upon launching, our test program automatically logs in and joins a collaboration room hosted on a Jabber server. After joining, the script simulates a user who adds a new image every seven to ten seconds to an area taken at random among the 16 predefined areas of the document as specified by the DOM. Once the request is accepted and the lock confirmed, a CGI PHP script uploads a random picture to a common HTTP server. The picture is picked amongst a set of three pictures, with the following sizes and likelihoods of being chosen: Picture1, 35kB in size, 50% of the time, Picture2, 75kB in size, 33% of the time and Picture3, 200kB in size, 17% of the time. Once the picture is uploaded, the state of area is changed to reflect the addition of the new object and an unlock message is sent to peers. The following setup was used: the first client (User1) located in Gatineau (Québec), was connected to the Internet via a domestic DSL connection with a downstream capacity of 3.0 Mb/s and also hosted our HTTP server. The second client (User2) was connected to the Internet via the wireless network of the University of Ottawa. A home network of three computers connected to the Internet through a residential cable connection, hosted up to four clients: two PCs connected by wire (User3, User4 and User5); and a laptop connected over an 802.11b wireless connection (User7). Finally, a fourth computer hosting one user (User6) was connected to the Internet through a broadband cable connection in Ottawa. The Jabber server is hosted on jabber.org located in IOWA, US.

Request Delays

Figure 3.3 shows the average of delays to obtain a lock, and total time to make a modification to the document, as a function of the number of users present in the collaboration room. Figure 3.4 shows the per-request total time to make changes to the DOM throughout the life of the simulation, of around twenty-five minutes. It can be observed that the time to obtain a lock and to perform an update is relatively stable up to six users, with the majority of changes taking less than five seconds. The system requires up to three seconds on average (with seven participants) to confirm that the area is not locked by

somebody else, and a few more to actually perform the change. The addition of User7 increases both the time to obtain a lock and to perform an update. We have identified three factors that might affect performance in our experiment. First, most public servers, in order to remain reliable and fair to all, use *choking* to delay or drop messages from a user who is sending heavily. Second, it is expected that our script attempts to upload, on average, around 10 kB/s worth of data, per user. This implies an average of 40 kB/s in upload for the network hosting User3, User4, User5 and User7. Taking into account that another bandwidth-consuming application was running on one of the machines (a BitTorrent client) it seems reasonable to believe that the upload bandwidth became saturated. Finally, with two of our test machines running wireless connections, it is suspected that this might cause some of the observed spikes in delays.

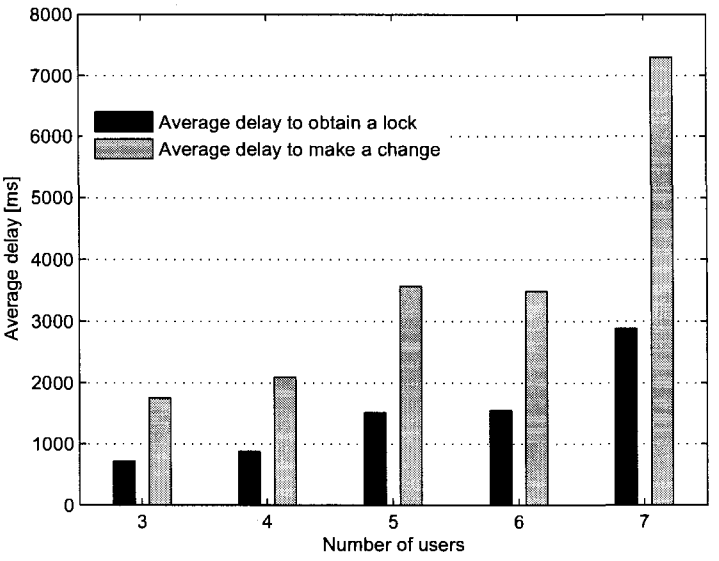

Figure 3.3: Average delays to obtain a lock and make a change

Request Results

The result of issued request is shown in Figure 3.5. Requests *not sent* are those that were canceled either because the area was already locked, or because the activities were paused in order to let a user join the collaboration room. *Canceled* requests are requests that were retracted when another user was logging in. *Timeout* requests are those for which

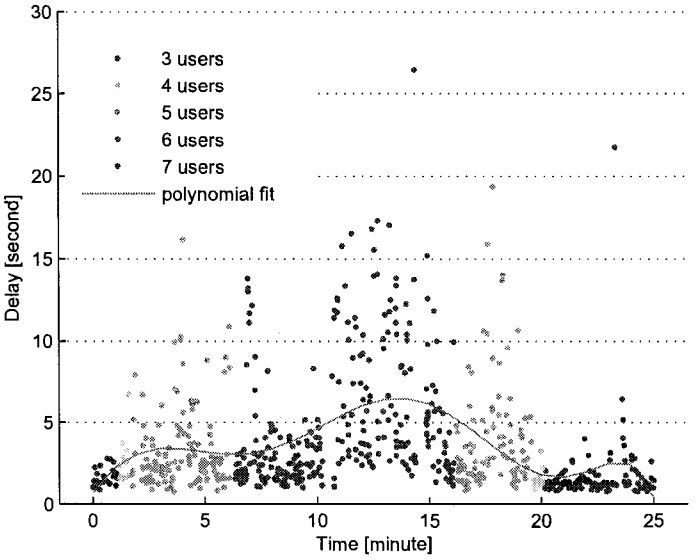

Figure 3.4: Time to make a change during the life of the simulation.

not all accept messages were received within five seconds; those were re-issued. *Denied* requests are those that were withdrawn because another peer attempted to lock the same area a moment before. *Successful* requests are those that completed successfully. We observe a constant decline in the proportion of successful requests when more users participate. One of the reason is that our script attempts to make changes to a randomly chosen area. The probability of picking the same area increases with the increasing number of users, in fact in case of 7 users and if users made requests at the same time, around $1 - \frac{16!}{(16-7)!}/16^7 \cong 78\%$ of requests would have at least one conflict. In real life, humans might act somewhat less randomly and make lesser amounts of conflicting updates. The increased traffic due to conflicting requests may explain the increased number of time-outs as well.

3.3 Bulk Data Transfer

In this section we address the bulk data transfer requirement of our collaborative multimedia application with JXTA protocols. JXTA is a set of protocols that specify basic functionality required by peer-to-peer type applications. Functions such as peer discov-

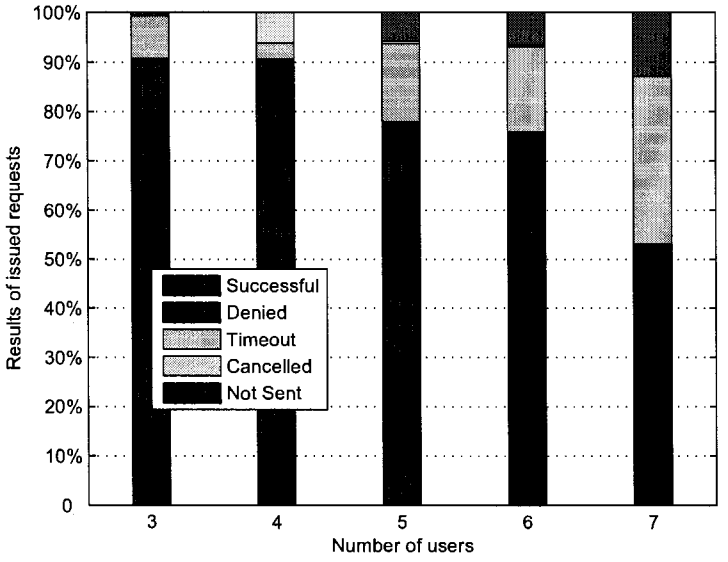

Figure 3.5: Results of issued requests as a function of number of users

ery, peer communication and their associated management are performed by publishing and exchanging XML advertisements and messages between peers. JXTA architecture provides a standard service discovery framework, allowing 3rd party services to be offered. CMS is an example of such a service.

The JXTA Content Management System (CMS) is a service used to manage and transfer shared files. Peers that wish to share a file with others generate a content advertisement. Peers that wish to download a specific content perform a search for an appropriate content advertisement and initiate the download. CMS implementation allows to download the same file from multiple sources, but requires the file to be fully downloaded before it can be offered to others (i.e. creating an advertisement for a partially-downloaded file is not supported). Although conceptually a request for the file is made, on a lower level, CMS handles requests and replies for the 64kB *chunks* that make up the whole file. Each request specifies the start and end of a range $(chunk_i, chunk_k)$ for $k > i$, which is sequential from the beginning of the file. A time-out is associated with each request, triggered if a *hole* has been found in the list of received chunks. When this happens, a request is issued for the missing range.

Results of a simulation where 7 peers download a file using CMS is presented in Figure 3.6. The download completes when all the pieces of the original file are received.

The graph shows the number of pieces received by each peer and the number of pieces transmitted by the source as a function of time, in units of 5 seconds. Since CMS allows to share a file only if it is complete, we observe that all participants connect to the only source offering the file. The communication between peers being TCP-based, the source divides its bandwidth more or less equally among all the peers. On the other hand, if the peers did not start at the same time we could expect different results.

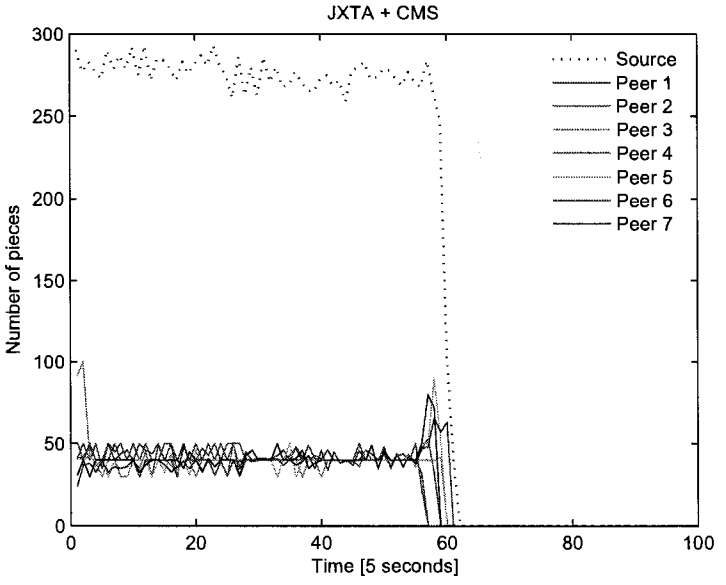

Figure 3.6: CMS file distribution performance

We can improve this performance profile by extending CMS to be more suitable in the case where all participants start downloading at the same time, by removing the two restrictions of fully downloading a file before sharing it, and of sequentially downloading the file from the beginning. The resulting Extended CMS (eCMS) allows the peers to share individual chunks (64kB) from an arbitrary start, and in any order. This flexibility allows us to define two different ways of completing the download, as described in the subsequent sections. Compared to CMS, eCMS uses different content advertisements, provides support for new messages, and its handling of issued and received requests for pieces (chunks) differs. As soon as a peer obtains a piece of the shared file, it will issue a content advertisement that contains the piece hash (for identification purposes) as well as the hash of the whole (parent) file. This ensures that peers interested in the file can discover other peers that are currently downloading that file as well. After joining such

a *swarm*, peers will exchange state messages and attempt to download pieces from each other. The state message provides a way for peers to exchange information necessary to decide which pieces are available for upload. Once a peer has obtained all the pieces of the original file, it will create a content advertisement for the complete file and publish it at the RV peer. Other peers may however still make requests based on the previously discovered content advertisement for a piece. A message will be sent to the requesting peer notifying it to use the new advertisement instead. The above changes ensure that peers can share incomplete files with other peers. Each peer will keep track of the pieces it currently owns (successfully downloaded), and of the pieces it has issued a request for. Pieces that are not part of either set and are available at another peer may be included in a request for pieces. The piece selection algorithm, the format of the request and interpretation of the request are modified in the eCMS. In our implementation, pieces are selected randomly, or based on a preassigned partition. The format of the request is similar in both cases, and contains a sequence of numbers $(i, ..., k)$ identifying individual pieces. These in turn, are interpreted by the receiving peer as a series of requests for ranges $(chunk_i, chunk_{i+1}), ..., (chunk_k, chunk_{k+1})$ where $i \neq k$. A time-out is associated with every piece, triggered if a piece has not been received within 20 seconds. That specific piece then becomes a candidate for another request.

3.3.1 Bulk Transfer Protocol

The purpose of the protocol is to allow peers to exchange information about pieces of a file, and to share them with other participants. In the next sections we describe how decisions are made about which piece to request, what messages are used to exchange information and pieces, and finally, the behavior of peers while exchanging messages.

Sharing Pieces

In the context of our multimedia authoring application, a single peer will possess a file that all other participants will want to download. Furthermore, this transfer process will be initiated roughly at the same time by all. Various approaches exist to assess the performance of such a system, including parametrization of the inter-peer behavior (peers that do not have a complete file), and simpler one, by studying the effects of changing the way the source peer (peer with the complete file) uploads pieces of the file to various peers. We describe the latter in the following discussion. We would like to maximize the availability of the pieces over all participants. This can be achieved by either peers

randomly selecting which required pieces to request, or by the source assigning equal and disjoint file partitions from which peers must request required pieces. As the download progresses, there may be multiple peers offering the same piece, and each peer must keep track of the pieces it already owns, $O_{wn}\{...\}$, and of the pieces currently requested but not downloaded yet, $D_{ownloading}\{...\}$). Each request for pieces is made from selecting some pieces from the set of remaining pieces, $R_{emain}\{...\}$ in order to create $F_{ull}\{f_1...f_{full}\}$. In the random case, a peer i will send a request for pieces to peer j by picking randomly up to 15 pieces from $R_i\{...\} \cap O_j\{...\}$, where j can be another peer or the source. In the partition case, a peer i will send a request for pieces to the source by picking sequentially up to 15 pieces from $R_i\{...\} \cap P_i\{f_n...f_m\}$, where $P_i\{f_n...f_m\}$ is the partition assigned to peer i by s. If $R_i\{...\} \cap P_i\{f_n...f_m\} = \varnothing$ then the peer picks sequentially up to 15 pieces from $R_i\{...\} \cap O_s\{...\}$. If the other peer is not the source, then peer i sends a request for pieces to peer j by picking sequentially up to 15 pieces from $R_i\{...\} \cap O_j\{...\}$. Partitions of equal size are assigned by the source peer on a first-come, first-served basis, and the source has prior knowledge of the number of participating peers.

Exchanging Messages

There are four types of messages that participants exchange to complete the download:

State Message. A request for the state can be sent explicitly, or embedded in the request for pieces. Each peer will explicitly request state information from a peer it is currently communicating with whenever this state information is set to null. State information will be set to null at the initiation of the download, and whenever a request for pieces cannot be created due to lack of available pieces. This message is sent at the beginning and when selecting a suitable peer to download from. State request is also embedded (or piggybacked) in a request for pieces whenever the next request for pieces cannot be created due to lack of available pieces. This method of requesting state information is used whenever a peer is actively downloading from another peer. State request is composed of the message name, and in the case of an explicit state request, the requesting peer JXTA identification. State reply will be sent whenever a request for state is received. Each peer keeps a state bit array representing all the pieces of the file it currently can share with other peers. The maximum size of the bit array is less than 2400 bits, or 300 bytes (this is dependent on the size of the file to transfer and the size of the piece). This information, plus message name and responding peer JXTA identification form the payload of the state reply.

Piece Message. A request for pieces contains a numerical identification of requested

pieces. Up to 15 pieces can be requested at a time, and the message contains the requesting peer JXTA identification. Replies to this request are handled by the underlying JXTA/CMS implementation, and are sent containing individual pieces (64kB chunks).

Complete Message. This message is rare, and its purpose is to notify that a peer has all the pieces. Other peers use it to stop sending state request messages to that peer.

Partition Message. In the partition case, two extra messages are exchanged between peers and the source only, the request and reply for a partition. These messages are exchanged prior to any piece requests. The purpose of these messages is to allocate a specific partition to each peer.

Peer Discovery and Behavior

A rendez-vous (RV) peer is used to facilitate the startup phase of the collaboration. All participants send a message to the RV peer advertising their presence (JXTA advertisement) and wait to discover 8 adverts (7 peers and 1 source) before proceeding with the actual file transfer. This ensures all peers start roughly at the same time. Only the source advertised that it has a file to share (CMS advertisement), and therefore all the peers attempt to connect to the source and request one piece of the file. Once this piece is received, a peer publishes its advertisement at the RV peer that it has a file to share. All the peers (except the source) attempt to discover new peers offering to share the file every 10 seconds. Once other sources of the file are discovered, peers start to behave as client and server with respect to each other.

Peers behave as *client* when requesting pieces from other participants, and as *server* when fulfilling requests for pieces from other participants. Server behavior is provided by the underlying JXTA/CMS implementation (with the slight modification to handle state messages), and is therefore the same for all participants. Since the source has the complete file, it never behaves as a client and it never receives requests for its state information. Other participants implement exactly the same client behavior. They select up to 4 peers to connect to, chosen from a randomized list of discovered peers. A connected peer is dropped in favor of another peer if it fails to upload pieces in three attempts, either by not having any pieces to upload or if a requested piece is not received within 20 seconds. Upon receiving all the pieces to reconstruct the original file, a client updates its content advertisement and becomes a source.

3.3.2 Setup and Experiments

As mentioned previously, the performance of this protocol is evaluated without integrating it with the Jabber-based protocols. The setup consists of 3 bridged LANs, the first providing 100Mbps supports 4 PCs (a.k.a fast peers), the source (peer with the complete file) and the RV peer, the second providing 100Mbps supports 1 fast peer, while the last providing 10Mbps supports 2 PCs (a.k.a slow peers). We ran 5 tests for both the random and partition cases and we collected logs for each machine after each test. We use arithmetic averages over those tests in all cases, unless otherwise specified. The size of the transferred file is 152.5MB, each piece is 64kB, and thus we have 2383 pieces.

Time Required to Complete Download

The time necessary to complete the download is a primary metric. The download completes when all the pieces of the original file are received. Instead of showing the cumulative number of pieces received as a function of time, the graphs in Figures 3.7 and 3.8 show the number of pieces received by each peer as a function of time, in units of 5 seconds. Peers request pieces based on the information about which pieces a participant can provide. This information is obtained by requesting a state message. The number of state messages sent as a function of time is shown on the graphs as well.

Participants with fast connections complete downloading in less than 200 seconds (with most finishing around 130 seconds). Participants with slower connections (Peer 3 and 7), complete downloading in less than 450 seconds. The time necessary to complete downloading is roughly the same in both the random and partition cases, although most of the fast peers seem to finish around 20 seconds faster in the random case than in the partition case. Slow peers on the other hand seem to finish around 20 seconds faster in the partition case than in the random case. We have also ran test with only the source and one fast peer, yielding 97 and 81 seconds for random and partition cases respectively. Due to the limited nature of the experiments however, no definitive inference can be drawn, and further study is necessary. In both cases, peers start by downloading from the source, and around 30 seconds later obtain advertisements that allow them to start sharing pieces. This is an artifact of our implementation, where JXTA discovery happens periodically every 10 seconds. The period when peers download only from the source could be reduced by reducing this discovery period. The number of packets peers can obtain from the source will be a function of their own connection speed, and of how many peers the source is serving. After a peer discovers other sources of pieces, the

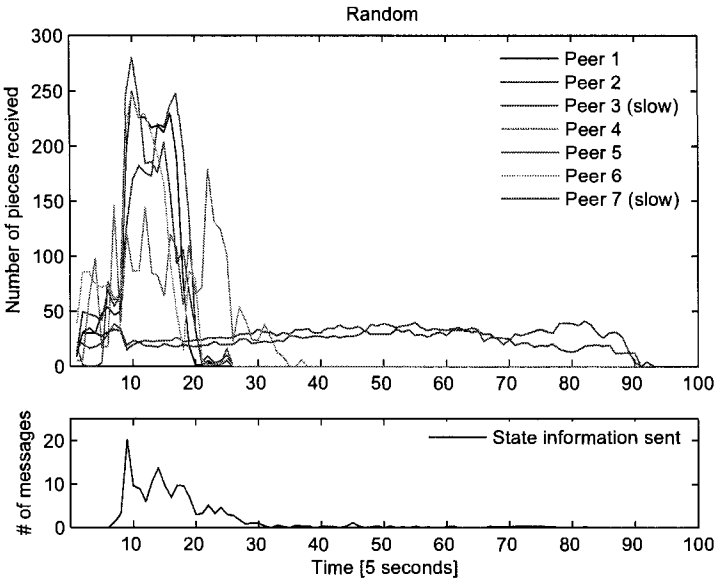

Figure 3.7: Number of pieces received by peers (random)

number of pieces received increases for the fast peers and drops for the slow peers. This is expected since fast peers have more download bandwidth than is available to them from the source only, whereas slow peers must now share their limited bandwidth with other peers. In this case, slow peers cannot improve their performance by connecting to multiple peers, and should limit their download connections to less than 4 currently allowed in the simulation. In order to minimize downloading time for all participants whenever there is more fast peers relative to slow peers, slow peers should generally behave as clients only, although the exact balance for which this holds needs further study.

Load on the Source and Users

We also look at the sustained load by the source and peers, expressed as the number of pieces transmitted as a function of time, in units of 5 seconds. The sum of the peers' contribution is shown instead of the load on individual peers. This gives us a comparison of the work performed by the source relative to the work performed by other participants and is shown in Figures 3.9 and 3.10.

We observe that the source provides 6793 and 9268 pieces while the peers provide

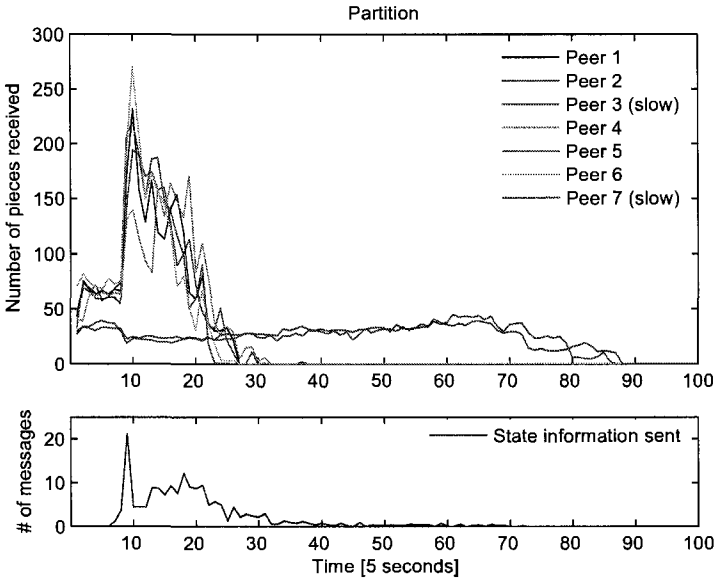

Figure 3.8: Number of pieces received by peers (partition)

10363 and 7840 pieces in random and partition cases respectively. In both cases the number of the pieces transmitted is greater than required for 7 copies (17156 and 17108 against 16681) due to duplicate pieces. In both cases after around 100 seconds, the number of pieces transmitted by the source and peers decreases sharply. In the random case, the contribution by individual peers is roughly equal to the source's contribution reflecting the fact that pieces are by that time widely available. In the partition case however, the source provides relatively more pieces than the peers up until around 200 seconds. Additionally, participants rely relatively more on the source than other peers for pieces, implying that if the source is located on a slow peer, the partition case may cause slower download time than random case.

Cost of Sharing

Next, we assess whether there is a relationship between providing pieces (behaving as a server with respect to other peers) and download time (time to receive all the required pieces). The scatter diagram in Figure 3.11 presents results for each trial (averages are not used), plotting download time as a function of number of pieces uploaded. To highlight the relationship, linear and quadratic fitting is used for slow and fast peers

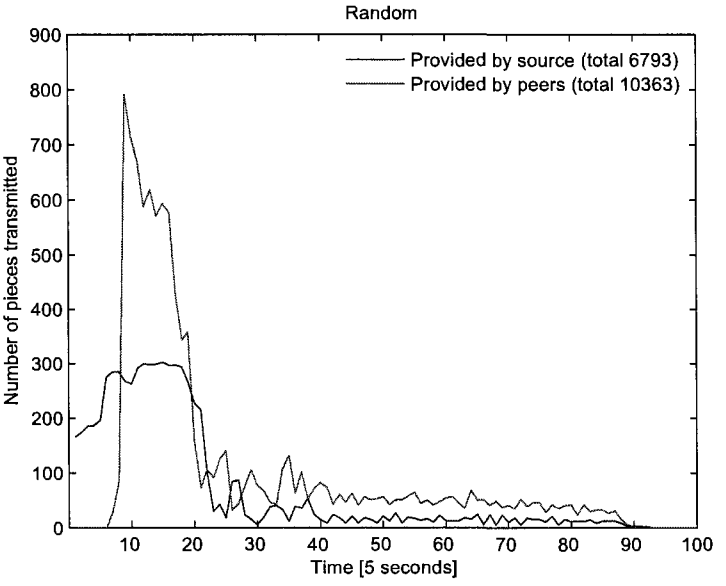

Figure 3.9: Number of pieces transmitted (random)

respectively.

Due to the wide bandwidth difference, slow peers fulfilling requests from fast peers will have a marginal effect on the fast peer's performance, but at a substantial cost to themselves. This is clearly shown in the diagram where increasing number of shared pieces increases the time necessary to complete the download. The trend for fast peers is more subtle, and shows a slight decrease of download time as the number of pieces transmitted increases, up to around 2500 pieces where the trend flattens (download time remains constant with increasing number of pieces). The initial decrease can be explained by the fact that by providing less pieces a peer removes upload bandwidth from the system, which increases the download time for all participants. Similarly, the flattening of the curve is consistent with the fact that once a peer finishes downloading the entire file, it starts to behave as a source to other peers. Finally, comparing the random and partition cases reveals broadly the same trends. If we design for the smallest download time for all peers, this would imply that the fast peers will share at least around 2500 pieces (close to the size of the file to share), while the slow peers should not share at all. This optimal design will most probably change with different proportions of fast to slow peers, as well as increasing heterogeneity of peers' available connection speed.

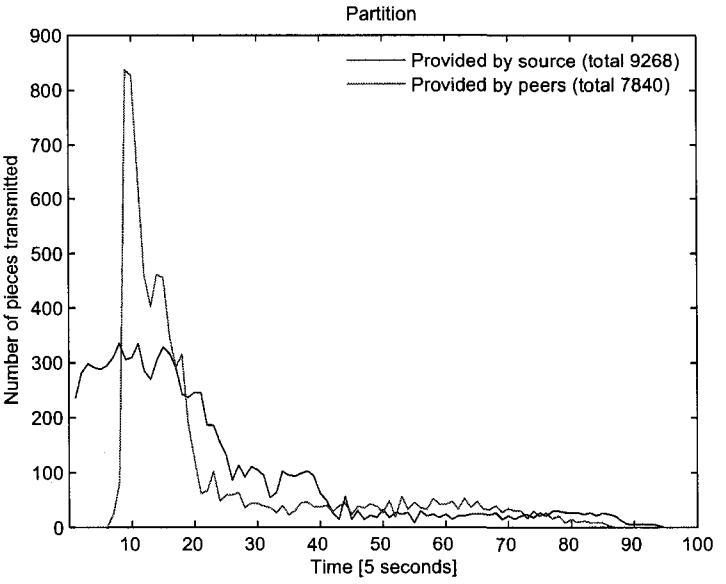

Figure 3.10: Number of pieces transmitted (partition)

Impact of Duplicate Pieces

In order to assess the effect of duplicate pieces on the performance, the scatter diagram in Figure 3.12 presents results for each trial (averages are not used), plotting the number of duplicate pieces received as a function of download time. Peers keep track of duplicate pieces only when they do not have the entire file, and ignore (and do not log) any pieces that arrive after it has completed the download. Whenever a requested piece is not downloaded before its timeout occurs, another request will be issued for the same piece. The peer therefore can download multiple copies of the same piece, and only the first downloaded copy is not marked as duplicate. The subsequent request may be issued immediately or at a later time - depending whether the peer requests pieces sequentially or randomly, and if one peer was replaced by another in the list of connected peers. We note that 24 duplicate pieces constitute around 1% of the total file, and that on average peers do not exceed that number. Any impact due to duplicate pieces is therefore bound to be small on download time. Effectively, the fast peers' download time does not show any correlation with the number of duplicate pieces received, while slow peers show only a slight correlation. More interestingly, we note that in the random case, fast peers have more duplicate packets than slow peers (average of around 11 and 6 respectively) while

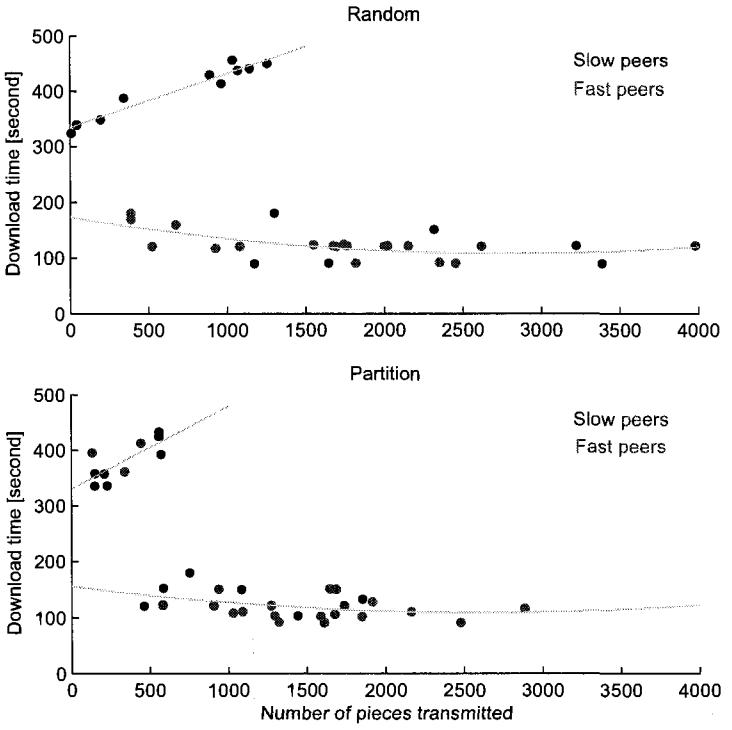

Figure 3.11: Number of pieces transmitted vs. download time

in the partition case, this is reversed (average of 2 and 26 respectively). It is expected that this could have a more important impact on the performance if we were to change the time-out parameter for each piece from its current 20 seconds.

3.4 Security

In this section we examine the framework from the security point of view. We carry out a brief risk analysis in order to highlight the security issues and present potential solutions. Using those solutions enables us to provide secured channels and to ensure that peers are properly authenticated. The objective of this framework is to enable easily deployable and upgradeable multimedia collaborative authoring application and as a consequence, any security solution should be as transparent to the users as possible.

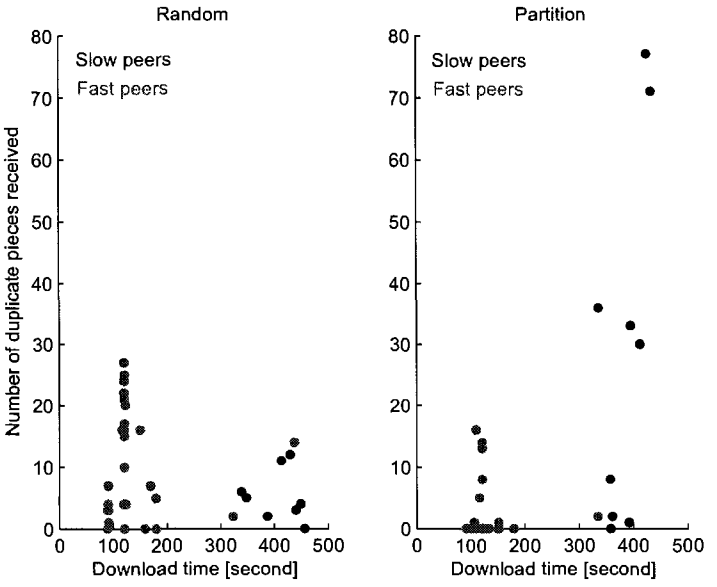

Figure 3.12: Number of duplicate pieces received vs. download time

3.4.1 Risk Analysis

We envision that the context in which our application is to be used is consistent with the following assumptions. First, we assume that the participants know each other; that there is a limit on the number of users; and that they are both reliable and genuinely willing to cooperate. Second, the system is not designed to support collaboration on highly confidential information, however access to shared objects should be access-controlled. Third, we assume that Jabber messages are always received within a finite amount of time.

Threats can be divided into accidental events and malicious actions. In our context, accidental events are for example problems related to the network state and users' computers and as such are out of our scope. The following malicious actions are identified: (1) eavesdropping on Jabber communication, (2) eavesdropping on file transfer service, (3) illegal access to the system that stores the shared objects and (4) illegal access to the chat room which would enable the attacker to disturb the service by sending misleading messages.

The sensitive data and information are: (a) the Jabber messages used for data consistency maintenance (integrity), (b) shared objects should not be modified outside of

the valid execution of the protocol (integrity), and (c) the data transfer corresponding to the upload/download actions of the shared objects (confidentiality).

The framework thus is open to the following risks: (i) an attacker could issue Jabber messages to some users in order to compromise the data consistency maintenance, (ii) an attacker could enter the chat room and transmit misleading messages with the same objective as before, (iii) an attacker gains access to the server that stores the shared objects and succeeds in downloading or modifying them, (iv) an attacker eavesdrops on the upload or download of a shared object.

If a Web server is used to ensure the bulk transfer as in Figure 3.2, the solution addressing risks (iii) and (iv) is well-known and is composed of HTTPS or SHTTP to avoid eavesdropping during data transfer, and a login/password mechanism to control the access to the server.

In the following sections we discuss the security features provided by Jabber and JXTA that can be leveraged in order to protect the framework from the identified risks.

3.4.2 Jabber Security

The XMPP Core protocol used by Jabber specifies how to achieve both communication confidentiality and mutual authentication between a client and a server and between two servers. Confidentiality is provided with Transport Security Layer (TSL) encryption while peer's authentication is achieved through the use of the Simple Authentication and Security Layer mechanism (SASL). If TSL and SASL are enforced, all Jabber packets are encrypted and thus it cancels the risk (i) identified in section 3.4.1. These two protocols can easily be enforced in our application transparently to the users, however TSL and SASL are optional for servers. This implies that users should either find or install and manage their own TSL and SASL enabled server. An end-to-end encryption for confidentiality and integrity independent of the servers is discussed in [3], but no implementation has been proposed. Jabber protocols support creating password-protected or members-only chat rooms. In both cases one of the members is in charge of creating a room. In the first case, this user will transmit the password to other participants, while in the second case a member *whitelist* has to be communicated to the Jabber server. Assuming SASL usage for user authentication, the access control over the room enables protection against risk (ii).

3.4.3 JXTA Security

We focus on two security mechanisms provided by JXTA: the peer group services and the TSL transport binding. One of the objectives of the creation of peer groups is to implement secured environment in which confidential documents can be shared. Furthermore a specific *Membership service* is used to grant or revoke access to members. JXTA implements TSL and so is able to provide confidentiality between peers. These two mechanisms address risks (iii) and (iv) identified in section 3.4.1.

3.4.4 Discussion

We showed in this section that some simple built-in security mechanisms can be leveraged in order to greatly reduce the risks associated with the infrastructure. Since most of the security mechanisms are transparent for the members and do not generate any significant overhead, the collaboration remains simple from the users' point of view.

3.5 Comparable Work

Related work was covered in Section 2.3 form the perspective of Collaborative Environments. Here, some comments specific to implementation of collaborative applications are included.

3.5.1 Collaborative Applications

As part of a broad collaborative environments [126] research category, the present research can be compared with many tools and frameworks. Collaboration on multimedia presentations is described in [138], but it requires an application-specific TCP-based protocol and places the emphasis on the authoring tool that is subsequently interfaced with communication and concurrency control modules. Collaborative authoring is addressed in [71] which introduces a Web Distributed Authoring and Versioning protocol (WebDAV) extensions enabling shared awareness through event notification. A framework for collaborative applications over a JXTA is presented in [78] and it provides supplemental services such as synchronous message transport, room administration, peer communication support and application space management.

3.5.2 Data Consistency

Data consistency is achieved through some kind of concurrency control implementation [22]. Implementations fall into two broad categories, optimistic and pessimistic. The first case assumes that no conflicts occur (hence optimistic) but detects conflicts at commit time and aborts conflicting transactions. Emphasis in those implementations is to reduce wasted effort as shown in [8]. The second case ensures that conflicts do not occur, and is usually based on some implementation of a two-phase locking mechanism [142]. Locking of course can lead to deadlocks, and is a major concern in those situations.

Implementations can also be centralized, distributed or token-based. Due to its nature, centralization is relatively easy to implement. Distributed implementations have to contend with exchange of many messages, group membership management and failures [30]. Finally, token-based implementations lessen the number of exchanged messages but need to consider token recovery and usually involve complicated implementations [48].

Locking mechanisms depend on either time synchronization or logical clocks implementation. Time synchronization can be achieved through algorithms such as presented in [91, 97]. Logical clocks [83] on the other hand, allow participants to agree on the order in which requests occur, not on the time they happened. Such implementations gracefully handle high rate of occurrence, as well as, small execution time of events. The choice between time synchronization and logical clocks depends on the exact requirements as to level of synchronization and implementation complexity.

3.5.3 Group Communication and Event Notification

A large body of work discusses alternative solutions to Jabber-supported event notification. An infrastructure presented in [11] supports development of scalable applications composed of loosely-coupled services that communicate via XML document exchange. Authors report the service invocation overhead to be roughly equivalent to that of a light-weight RPC protocol, and over an order of magnitude less than XML-based protocols such as SOAP/HTTP. An application-level multicast framework proposed in [112] targets group communication in interactive applications such as real-time collaboration by specifying a stateless mechanism - application level header contains an encoding of IP addresses of all multicast group members - together with a corresponding tree building algorithm.

3.5.4 Data Bulk Transfer

Bulk transfer is usually the domain of content distribution networks (CDNs). Recent studies suggest that users should share their upload bandwidth in order to help other participants get the file faster. Two technologies in particular, Application Layer Multicast (ALM) and implementation of a BitTorrent (BT) protocol make upload bandwidth utilization possible.

ALM allows for multicasting data at the application as opposed to the network layer, and can be separated into end-host and infrastructure solutions. The end-host solution involves hosts sharing the responsibility for forwarding data to others by organizing themselves into an overlay tree [109]. The infrastructure solution assumes that the infrastructure itself does the heavy lifting of efficient multicast while individual members of the infrastructure also act as proxies for the participants [158].

A BitTorrent file distribution usually consists of an ordinary web server, a *torrent* file, an active and centralized component, the *tracker*, an initial client with the full copy of the file to share and a set of downloading clients. A new client starts by downloading a torrent file containing the IP address of the tracker from a website indexing such files. The tracker keeps information about the peers that are currently active and acts as a rendez-vous point for all the clients of the torrent. Active clients periodically report their state to the tracker or when joining or leaving the torrent. Upon joining the torrent, a new client receives from the tracker a list of randomly selected active peers to connect to. Clients involved in a torrent cooperate to replicate the file by exchanging chunks of the file with one another. BitTorrent specifies two main algorithms, the *choke* and *rarest-first*, to make the process of chunk exchange efficient [25]. The choke algorithm encourages cooperation among peers, limits the number of peers a client concurrently exchanges chunks with and selects the best peer (optimistic unchoke), while the rarest-first algorithm controls which pieces a client will actually request in the set of pieces currently available for download.

While using the BitTorrent protocol in the context of our multimedia authoring application is possible, three issues would need to be addressed. BitTorrent is designed to efficiently replicate a file between a large number of peers that do not necessarily know each other, and in most likelihood will not cooperate on downloading another file again. In contrast, during a multimedia authoring session the same peers might collaborate on transferring multiple files. Torrent file creation and distribution would thus impose unnecessary costs if not adapted to our application. Secondly, the choking algorithm's

benefits - reciprocity and peer selection - will not be fully utilized. In our application peers are assumed to be fully cooperative and thus there is no need to implement incentives to trade pieces. Similarly, due to the limited number of participants, searching for a better peer could again impose unnecessary costs. Finally, the tracker provided services such as keeping a global view of the system, are not used in our case since they are designed to support interaction between a large number of peers.

3.5.5 Security Issues

The motivation for addressing security issues in SOA-based applications is presented in [49]. A general view on security is presented in [148] which investigates security issues from both the network and application layer perspective in peer-to-peer networks, and [41] which discusses security requirements in service-oriented architectures as applied to ubiquitous computing.

Applications built using Web Services protocols would naturally gravitate toward WS-Security specification (SOAP Message Security) which provides mechanisms for securing exchanges of SOAP messages [4] and emerging specification such as WS-Trust in order to provide a framework for security tokens and to broker trust relationships between participants. The Web Services Security Policy Language (WS-SecurityPolicy) [2] is based on WS-Policy and enables the specification of security constraints such as for example restricting Web Service access to only SOAP messages signed with X.509 certificates. In [26] authors argue that the policies specified using WS-SecurityPolicy correspond to low level security enforcement choices, and that a higher level policy language should be defined. They suggest a new language based on a formal semantics and propose associated tools allowing them to prove the correctness of the policies. The type of security policies addressed focus on authorization data, certificates and password management.

3.6 Summary

A collaborative authoring application will require certain basic services which can be supplied by Jabber and JXTA frameworks. The security of the overall application will be in large part dictated by the the security features of those technologies.

Jabber was designed to provide near real-time structured information exchange and thus naturally provides the required group notification service. We are using this service,

in conjunction with the presence service to create a collaboration protocol by distributing the responsibility for document consistency to participating users. The implementation of this protocol mandates a strict document consistency, and presents a great scope for improvements by taking the application semantics into consideration. Future work includes running the experiments on a single high-bandwidth network and to reduce the probability of conflicting requests in order to obtain a measure of the protocols performance while in a controlled environment.

Finally, JXTA framework was designed to facilitate communication and collaboration between peers, and thus lends itself to provide the bulk data transfer service. Future work will address the following two issues. In case of heterogeneity of peers connection speed, better results can be achieved if certain peers do not share their bandwidth, or limit their upload rate. This suggests that deriving bounds for this optimum point and providing ways of achieving it, need to be explored. Secondly, we have highlighted two approaches that - by varying the way in which the content is downloaded from the original source - impacts participants performance. Further study is needed to determine which approach presents better characteristics, and if they are specific to particular cases. Additionally, a comparison of our multimedia authoring implementation to an implementation based on traditional web services standards could yield further insights.

A certain level of security in service-based collaborative applications is paramount to ensure its adoption by users. For this reason, we carried out a security analysis on our framework. We were able to show how to use the built-in features of the technologies we use to ensure a proper level of security. More specifically we can ensure the confidentiality and integrity of the communications by using adequate encryption solutions, and the authentication of the peers.

Chapter 4

Incentives Reference Framework

In contrast to a client-server model where servers provide services to clients, a peer-to-peer model (P2P) dictates that an entity should act both as a server and a client with respect to other entities. If providing a service entails costs, then the entity acting as a server should expect some form of compensation, or more generally, should have an incentive to offer this service. Incentive mechanisms are however neither fully understood, nor effectively deployed on existing popular P2P networks, with the result that many entities choose to act solely as clients (free-riders) [7].

In the context of service-based collaborative environments, which may be built on top of P2P architectures (see section 1.1), incentive mechanisms will therefore have an impact on deployed applications. A natural interest then is in the understanding of this impact, and in deriving a quantifying measure of its effect. A lack of common reference point leads to difficulty in comparing characteristics and performance of incentive mechanisms. In order to compare these characteristics, a reference framework is presented. This framework presents elements common to all incentive mechanisms, and choices available for their design.

4.1 Introduction

Although incentive mechanisms are well documented in business, economy and sociology literature [17], their application to peer-to-peer networks is relatively new. Issues such as improvement of cooperation using techniques from economics and social sciences, applicability of various market models and viability of different revenue models in peer-to-peer networks were recently addressed in [136, 129] and [70]. The fundamental difficulty

in deploying incentive mechanisms rests in ensuring that they are resilient to failures and robust against malicious entities [52]. Since entities are expected to be independent and rational, there is no guarantee that they will adhere to a given protocol or mandated strategy [134, 103]. Ensuring that an incentive mechanism is not subverted is complicated by the nature of current peer-to-peer networks, where an entity can join a network with an easily obtainable identity [94, 55], and [45].

Some file-sharing peer-to-peer networks do not deploy incentive mechanisms, and yet are very successful. It is natural then to ask if incentive mechanisms are truly necessary and not purely of academic interest. It can be argued that the success of file-sharing applications also depends on other factors besides the level of free-riding. The cost of providing services (files) on existing file-sharing networks is close to zero and the expected quality of service correspondingly low. Coupled with the compelling nature of the exchanged files, this creates a very useful network for entities to join. However, if the cost of providing services or the demanded quality of service rises, it is unlikely that a peer-to-peer network without an incentive mechanism can be successful.

Large numbers of free-riders on the peer-to-peer network bring up questions about the viability of the network in general, and about its usefulness to other participating entities in particular [82]. As argued in [144], benefits of peer-to-peer networks manifest themselves in massive diversity. When an entity free-rides, it contributes to the reduction of this diversity, and therefore to the decrease of usefulness of peer-to-peer networks [24]. Whenever rational decisions made by an individual entity negatively impact the utility of other entities, incentives (i.e. reward or punishment) become necessary to ensure that the usefulness of a peer-to-peer network does not diminish. Practical implications of incentive mechanisms thus relate to increasing collaboration and availability of services, and target various applications, e.g., file-sharing [92, 40], distributed storage [104] and streaming [63].

The rest of the chapter is structured as follows; section 4.2 describes the INCA framework; an evaluation of the framework based on proposed incentive mechanisms is presented in section 4.3; comparable work is discussed in 4.4 and a conclusion is presented in section 4.5.

4.2 INCA Reference Framework

4.2.1 Reference Framework Elements

All the incentive mechanisms have the following elements. Four of those elements are essential, and one is optional.

Motivation. There must be a clear motivation for entities to interact and collaborate. Usually, motivation is provided by a prospective gain for an entity from joining and participating in a network. Since entities produce and consume services, motivation needs to be assessed from both consuming and producing perspective. This element is therefore concerned with defining the motivation for entities to participate in a peer-to-peer network.

Differentiation. For an incentive mechanism to function properly there must be a concept of differentiation, either of services, quality of service or between entities. Just as we can select a bank with the best terms for a line of credit, and banks can choose to whom and at what rate they extend a line of credit, participants in a peer-to-peer network must be capable of observing and acting upon perceived differences. This element is therefore concerned with the manner in which differentiation is defined and made accessible to participating entities.

Peer Selection. The goal of this element is the selection of an entity (or entities) most likely to lead to a successful interaction. In certain applications it is possible, indeed desirable, to obtain services from multiple entities concurrently, or to dynamically seek out better entities to interact with. In those cases, this element acts as "glue" for the differentiation and utility calculation elements. In other cases, it might not be necessary to include this element in incentive mechanisms.

Utility Calculation. Defining an entity's perception of derived utility from producing or consuming services is one of the fundamental issues for an incentive system. It is however possible to come up with potentially infinite number of definitions since the exact utility derived by an entity can depend on targeted application, various assumptions about available knowledge or conditions, or even intangible factors. This element is therefore concerned with the characteristics of objective functions rather than their specific definitions.

Metric Scope. Some parameters are necessary for the preceding element. Incentive mechanisms may choose various parameters as metric, depending on the target application or availability of variables. Combining different parameters into a composite

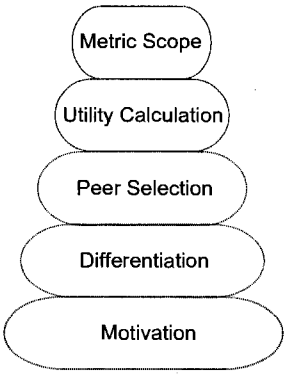

Figure 4.1: Incentives reference framework elements

parameter is also possible. The metric can also be thought of as a "proxy", or store of value, for services rendered or consumed. The main concern of this element is the scope of the metric within the peer-to-peer network, and not its specific definition.

Elements describing characteristics of incentive mechanisms are presented in Figure 4.1. They are represented as layered one on top of another to highlight their interdependency.

4.2.2 Motivation

Understanding why an entity should interact and collaborate with another entity is paramount to building effective incentive mechanisms.

Two popular file-sharing peer-to-peer networks, Kazaa [87] and BitTorrent [39] allow users to exchange files. Their protocols for doing so are different, but in both cases we can think of the offered service as the permission to upload contents from a storage device using some (preset) bandwidth. In the first network, an entity can access offered services while not offering any services itself. An attempt to distinguish between classes of entities based on their participation level was defeated by the release of a software client implementation (e.g. KazaaLite) which always reported its participation level as the maximum defined. In the second network, entities are rewarded with faster download speed in proportion to how fast they allow other entities to upload from them. An entity however can easily receive more services than it actually produces by disconnecting from the network after receiving the requested file.

It can be argued that these two networks do not have robust enough incentive mecha-

nisms, and a quick look at supply and demand curves illustrates the problem. Considering the price of consuming a service to be close to zero, the demand for this service can grow arbitrarily high. Since the price of supplying a service remains constant regardless of the quantity supplied, the equilibrium point would then be at an arbitrarily high level of the quantity of service. Assuming that the number of entities willing to supply a given service for free is limited, a gap between the quantity of service supplied and demanded will result. The net effect is that a number of consumers will not be able to fulfill their need for a service. Furthermore, every time a service is provisioned, the server entity incurs a cost that decreases its ability to keep offering services. If the peer-to-peer network does not continuously add such producer entities, then the network's capacity to offer services also decreases. It then follows that each entity consuming a service reduces the utility of the network to other consumer entities. A situation where there is a mismatch between supply and demand, and where each fulfilled demand reduces the utility of all, but where all consumer entities are marginally better in the short-term by consuming as much as they can, will potentially lead to the "tragedy of the commons", or in other words, the collapse of the peer-to-peer network. If such a drastic outcome does not come to pass, we are still faced with the issue of entities consuming more of the free services than their "fair share", or by choosing to restrict their contribution to the network, to avoid contributing their fair share of the cost. Such entities would in fact get a free-ride on the peer-to-peer network.

In effective incentive mechanisms, an entity should expect to either give or receive some form of compensation. Producers willing to offer services for free do exist, for example as a promotional give-away or as a loss leader strategy. Similarly, entities consuming a free service may decide to make a donation, in effect buying a service from a producer. Those examples however remain an exception rather than the rule, and entities will consume services for free if they can, while producers would prefer a more predictable revenue stream.

An obvious form of compensation is an exchange of money for the service, just as in the case of cash or credit card usage. Payment systems assume the existence of a currency that can be used outside of the network. Participating peers then have a motivation to offer services in order to receive this currency. If entities are capable of making properly informed choices in deciding on the appropriate level for the price of a service, then free riding should not exist. Such systems assume the existence of a robust framework for payment processing, which in peer-to-peer networks presents issues of its own such as, for example, reliable accounting mechanisms [66]. An example of an implementation based

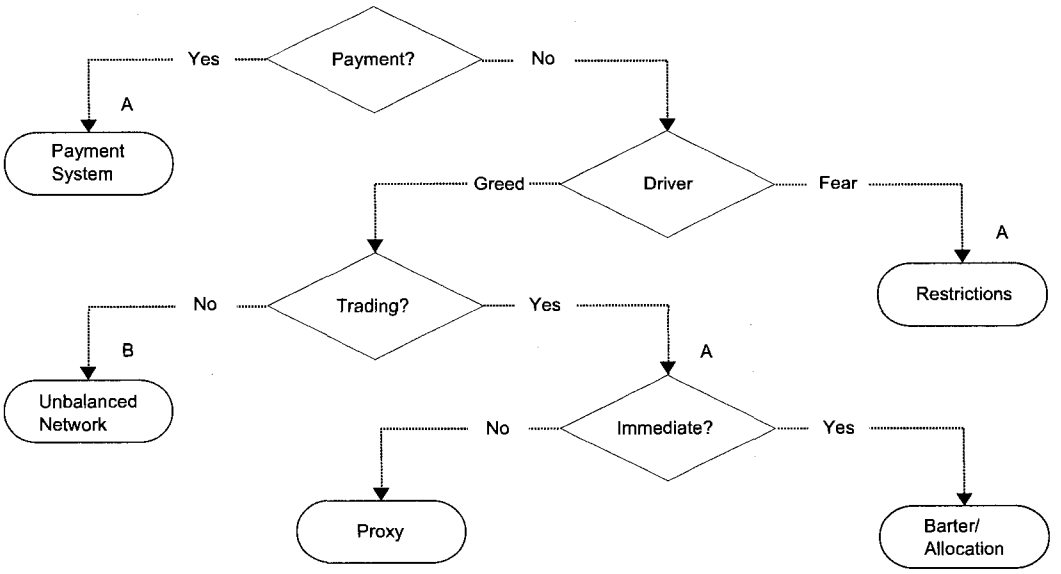

Figure 4.2: Schema of the motivation element

on a payment system is outlined in [16].

If there is no payment system, then entities can be motivated to participate in a network by either "greed" or "fear", or possibly a combination of both. On one hand, an entity may be enticed to contribute to the network by threats of sanctions and restrictions. Methods in this branch would work best in the context of networks where membership is restricted and participants have a well-known identity, such as communities or clubs. Ideas outlined in [130] could be applied to a potentially open and anonymous membership, and are based on client puzzles that participants might be asked to solve in exchange for services. On the other hand, an entity's intrinsic need or desire (eagerness) to obtain a service may be leveraged. Methods in this branch can assume that participating entities freely decide on the level of their participation. An entity might be eager to, for example, obtain some files from a file-sharing network, or store its data on the network in such manner that it is both secure and highly available.

Another form of compensation can be based on trade, or payment-in-kind. Entities trade if they allow access to each other's storage, bandwidth or computing power. A producer node can be compensated with another service or a promise of a service in the future. In both cases, entities are required to assign a value to their services, especially if these are not heterogeneous. In the first case, entities barter (negotiate) or allocate

some portion of their capacity to perform a service for other services, while in the second case a producer entity obtains a voucher redeemable at a later time from the network or the consuming entity. Such vouchers must be worthless outside of the network. The motivation element can therefore be based on some concept of proxy currency, on barter or some combination of those characteristics. If there is no trading between entities, or in some cases trading takes place but is inefficient, a situation allowing extensive free-ride exists, creating an unbalanced network.

Whenever there is potential to receive something tangible, such as money or service, participating entities have a motivation to collaborate. Figure 4.2 presents key design choices for the motivation element of incentive mechanisms. Points "A" is where an entity can be expected to be motivated to collaborate in a network. Point "B" illustrates two common problems, tragedy of the commons and free-riding.

4.2.3 Differentiation

The ability of a network to provide differentiation of services, quality of service or between entities is necessary for designing viable incentive mechanisms. The concept of differentiation is not dependent on what exactly is chosen as basis to assess differences. Without loss of generality then, the following discussion will use quality of service as example.

Two main choices exist for differentiation. The first, already familiar to most shoppers, places the burden of describing a service and its cost on producers. Consumers then need to understand what is offered, and how to compare offers. The second involves producers calculating what class of service to offer to consumers. These calculations can be based on factors such as credentials supplied by consumer, the result of optimization of an objective function, or a combination of those two methods. For example, in the case of a customer with a bad credit history, a bank might decide to extend a line of credit at a punishing interest rate, if at all.

In the case of the first choice, we can further analyze this element based on which entity describes a particular service. In the simplest form, an entity is responsible for making its own claim. Claims can also be made by a third-party, which can be trusted to varying degrees. If the third-party is not fully trusted, incentive mechanisms can rely on referral or reputation infrastructures [141, 150, 135, 43]. If the third-party is fully trusted, design of incentive mechanisms becomes easier but at the expense of shifting complexity to the design and deployment of the trusted third-party itself.

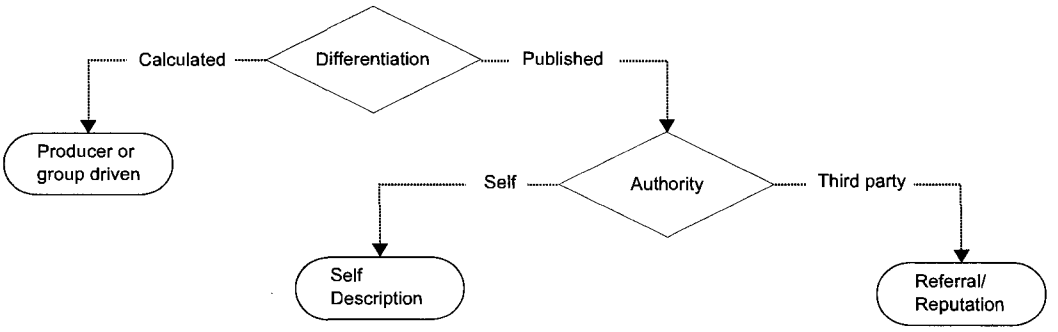

Figure 4.3: Schema of the differentiation element

The second choice implements an explicit reward and punishment mechanism. Producers can make decisions based either on their personal goals, or on some common group-specific goals. In the latter case, a group of producers ensures similarity of service offered to comparable consumers. Since the consumer does not know a priori what quality of service to expect, its natural behavior will be to try different producers and keep the best one.

Figure 4.3 presents key design choices for the differentiation element. By introducing the notion of receiving something tangible in compensation for services in the preceding layer, we automatically require the concept of differentiation in an incentive system. This element therefore describes the manner in which differentiation is defined and made accessible in the network.

4.2.4 Peer Selection

Peer selection element is the glue that holds the differentiation and utility calculation elements together, and indeed, it is sometimes subsumed by one of those neighbors. Basic design choices of peer selection element are presented in Figure 4.4.

In simple cases where a consumer is presented with a choice of producers, peer selection reduces to selecting a producer that maximizes an objective function of the consumer. Similarly, if the differentiation element is defined by a probability function for a producer, the peer selection element reduces to the choice of serving or not serving a particular consumer.

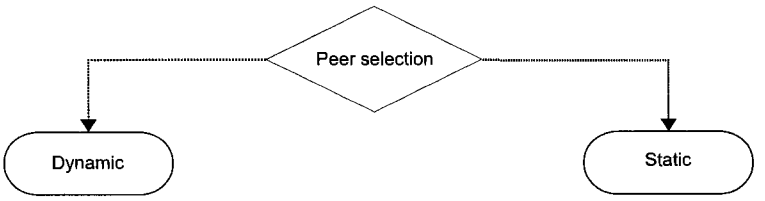

Figure 4.4: Schema of the peer selection element

Why is the peer selection element important then? Considering the transient nature of entities in a peer-to-peer network, we immediately see that some producers who are providing a service to consumers can leave the network, or new producers with possibly better qualifications can join the network. It would be beneficial for consumers to be able to monitor and seek out better partners for interaction and collaboration. This is also the case when producers select consumers with whom to interact. In instances where a producer optimizes an objective function, it might obtain better results if it replaces some consumers by others.

The goal of this element is the selection of an entity most likely to lead to a successful interaction. The basic design choice defines whether an entity's interaction with other entities is static or dynamic.

4.2.5 Utility Calculation

Selection of a utility function is one of the fundamental decisions for an incentive mechanism as it will influence all interaction and collaboration between entities. Each entity could define its own utility function, but for the sake of clarity, we will assume that such functions will be defined in terms of specific applications' requirements, or based on well defined goals. In [85], for example, this function is defined in terms of cost and requirements as specified by parallel downloading or streaming of audio/video objects. In [21] on the other hand, this function tries to minimize the time necessary to select the best source for a file download.

Despite the potentially unlimited number of utility functions, the basic design choice of the utility calculation element is quite simple and is illustrated in Figure 4.5. Each entity will optimize an objective function that results in highest utility as perceived by itself, or by a group. In the former case, each entity is independently making rational decisions, and hence exemplifies a "free-for-all" concept. In the latter case, each entity

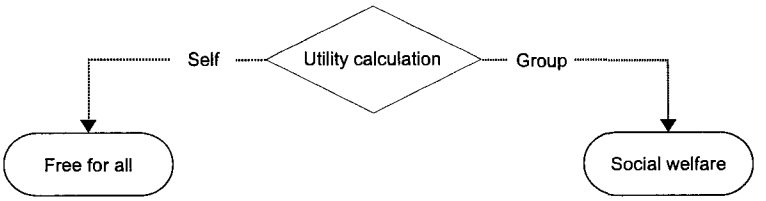

Figure 4.5: Schema of the utility calculation element

is making decisions that are rational in a collective context, and thus tries to optimize "social welfare" of the group. A group can be the whole or some subset of a peer-to-peer network.

This element, therefore, defines the characteristic of the chosen utility function, whether an entity explicitly acts in its own interest or the interest of some group.

4.2.6 Metric Scope

This element includes the definition of parameters used in the utility calculation element. For example, in the case of file-sharing or network storage applications entities can exchange information about their available upload bandwidth or available disk space. Quite often, it is very helpful to think of this layer as defining a proxy, or an intermediate store-of-value used in valuation of services exchanged between entities. This occurs for example in reputation frameworks, where an entity is judged by others depending on their reputation. Reputation itself can change depending on the amount and quality of supplied services. The basic design choice of this element is illustrated in Figure 4.6.

Although many metrics can be used, the basic characteristic of this element is the scope of such metrics within peer-to-peer networks. A group in a network can share the same metric, or each entity can create and use its own. Referral systems are examples of systems that need to carefully manage the semantics of metrics, while simple machine-learning implementations use metrics that have local meaning only.

If the chosen metric is an abstract concept such as *reputation* or *trust*, a corresponding model, associated data structures and operations need to be defined. Work presented in [5] assists users in identifying trustworthy entities, and gives them the ability to enhance their understanding of other entities' subjective recommendations. A method to assign each entity in a network a unique global trust value in a distributed and secure manner is presented in [75], and works by aggregating local trust scores of all entities, reflecting

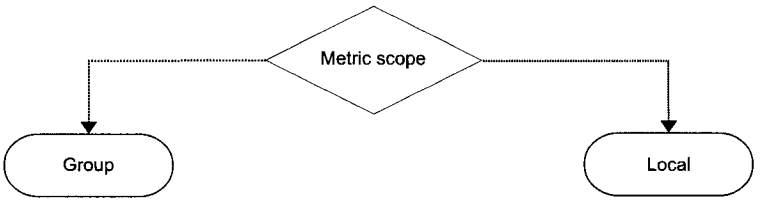

Figure 4.6: Schema of the metric scope element

their experience with a specific entity.

The main concern of this element is therefore the scope of the metric within the peer-to-peer network. In cases where a group of entities share a metric, this element may also be concerned with storing and managing metric information in a decentralized manner.

4.3 INCA Evaluation

Addition of incentive mechanisms to collaborative environments built on a peer-to-peer architecture has the potential to increase the robustness of various applications. To compare relative qualities of incentive-enabled systems many QoS parameters can be used, but ultimately, the level of satisfaction of a user will determine if a system is good or not [123]. Incentive mechanisms will impact the user's perception of the interaction in various ways. In order to start defining quantitative parameters, four questions can be asked:

- Does the user's benefit increase with increasing contribution?
- Do users with lower level of contribution receive lower benefit?
- Is a change in contribution reflected on the user's benefit in a timely manner?
- Is the received benefit stable if the contribution level is also stable?

In general, if the answer is negative to those questions, the incentive mechanism does not improve user's perception of the interaction. The following discussion presents four examples of implemented incentive mechanisms in different applications.

4.3.1 Game Theoretic Approach to File Sharing

Work presented in [92] strives to offer service differentiation dependent on the amount of each entity's contribution to the peer-to-peer network.

Motivation In this case, peers trade by means of a proxy, present production for future consumption. Consumers compete for resources provided by the producer. The producer allocates more resources to the consumer whose contribution level is higher relative to other competing consumers.

Differentiation Each entity has publicly available credentials supplied by a trusted third-party. Producers use these to calculate how many resources should be offered to a consumer, depending on the group currently being served. Producers' differentiation is not specifically addressed, but since their contribution levels and available bandwidth are known, consumers can presumably devise strategies that take this information into account.

Peer selection Peer selection is not discussed from consumers' perspective: it is ancillary to the differentiation element. Producers tolerate joining and leaving consumers, and stop interacting with a consumer that does not provide proof of service. Producers attempt to distribute their resources according to a specific formula among all consumers, but do not make any attempt to interact only with a subset of requesting consumers; as such producers have a static peer selection element: it is subordinate to the utility calculation element.

Utility calculation Consumers' utility calculation is not explicitly discussed. Instead, if consumers follow the given protocol (and have a non-zero contribution level), they can be guaranteed a share of resources as calculated by the producer. The producer computes consumers' utilities and tries to optimize for all connected consumers. By maximizing the sum of connected consumers' utilities, a producer guarantees to maximize its contribution level. A producer is in fact maximizing its contribution level by maximizing the utilities of connected peers.

The producer assumes that each connected consumer has the following utility function: $U_i(x_i) = log\left(\frac{x_i}{b_i} + 1\right)$ where $x_i \in [0, b_i]$ and is the allocated resource to consumer i, and b_i is the requested resource by consumer i. The producer then performs the following constrained optimization: $max \sum_{i=1}^{N} C_i U_i(x_i)$ subject to: $\sum_{i=1}^{N} x_i \leq W_A$, where $x_i \in [0, b_i] \forall i$, W_A is the total available resource at the producer and C_i is the contribution level of consumer i. The authors use a competition game model where C_i and W_{Ai} are known to all participants and prove that $b_i^* = \frac{W_A C_i}{\sum_{j=1}^{N} C_j}$ for $i = 1..., N$ is a Nash

equilibrium. Similarly, the authors prove that the resource allocation to consumers (x_i) maximizes the producer's contribution level.

Metric scope Metric is based on globally known quantities: contribution level of an entity (C_i), and the amount of bandwidth resource provided by a producer (W_{Ai}). The contribution level is calculated based on evidence of rendered service: digitally signed requests for a specific amount of resource from a consumer (b_i).

Analysis

Applying INCA framework allows us to highlight some further research areas for the preceding publication.

- The same consumer might be offered varying amounts of resources by different producers since at equilibrium $b_i^* = x_i^*$ for $i = 1..., N$ and which depends on the contribution level of all connected consumers to a particular producer offering W_A. It would be in the consumers' best interest to search for the most accommodating producer. This can lead to an impact on the overhead communication between entities.

- Producers do not necessarily need to maximize their contribution level, and can simply provide some resources to a consumer for its contribution level to go up. The resulting contribution level might be enough to satisfy an entity that subsequently wishes to consume resources (it is guaranteed not to be resource-starved). This is due to the fact that a producer does not know a consumer's true utility function, assuming $U_i(x_i) = log\left(\frac{x_i}{b_i} + 1\right)$ instead. If such a situation occurs and is persistent, then an entity, even conforming to all of the protocol's rules, cannot be guaranteed to receive its fair share of resources undermining the basic premise of the motivation element.

- If there is no concept of decay of the contribution level, producers can build it up and then free-ride, undermining the basic premise of the motivation element.

- Over the long term, and as contribution levels of entities increase, new participants might have to offer resources for a long time before they can successfully compete with more established consumers. This could be unfair and would undermine the basic premise of the motivation element.

- Producers are interested in maximizing their contribution level. Maximizing the sum of consumers' utilities is a more complicated way of just maximizing the usage of offered service: a producer might find it more expedient to provide resources to only one consumer if $b_i \geq W_A$ rather than calculating precise allocations to multiple consumers. By not following the prescribed mechanism, such producers would undermine the basic premise of the motivation element.

- Two entities can collude by sending evidence of an improbably high allocation of resources. An entity would then have a very high level of contribution since there is no way to verify or exclude that evidence. Such a security loophole would undermine the basic premise of the motivation element.

- Rare files vs. popular files: if an entity shares rare files - generating less demand for its resources - its contribution level might be smaller than if it were offering popular files. In order to increase its contribution level, a producer could switch from offering rare files to offering popular files. This can lead to an impact on the diversity offered by the network.

4.3.2 Incentives in Media Streaming

Work presented in [63] describes an incentive system for peer-to-peer media streaming applications. Based on the insight that random peer selection provides random quality, the authors propose a system in which collaborative nodes are rewarded with higher flexibility in peer selection.

Motivation The incentive system is based on participants' score which increases when providing a service. Nodes with a higher score have more flexibility when selecting a producer, which leads to higher streaming quality. Motivation is thus based on proxy trading.

Differentiation Entities publish their scores. In order to mitigate potential cheating and collusion, the authors suggest using reputation systems such as [52] or [75]. Producers also differentiate themselves by publishing their availability and offered bandwidth rates which are also required to predict the expected streaming quality.

Peer selection Consumers locate producers willing to provide them with a service. Those producers are classified as active and standby, and the consumer can select producers that best match its requirements at any time. Peer selection is therefore dynamic. Although not explicitly stated, peer selection appears to be dynamic at the producer as

well. A producer could for instance stop providing services to an existing consumer if a more suitable consumer made a request.

Utility calculation Entities choose their contribution level in order to maximize their own utility. The utility function is defined as $U_i(x_i) = a_i Q(x_i) - b_i C(x_i)$ where Q and C are the quality and cost functions dependent on the contribution level x. The constrained optimization is: $max\ U_i(x_i)$ subject to: $U_i \geq 0$ for $x_i \geq 0$ The cost function is based on bandwidth and storage usage $C(x_i) = (c_L + c_T)B_{out}$ where c_L, c_T are the unit storage and unit transmission costs respectively, while the quality function is defined as a monotonic non-decreasing in user score, asymptotically reaching a value of Q_{MAX} and having a non-negative initial value $Q_{BestEffort}$. User's contribution first gives a certain score. That score is then compared to the scores of other entities, yielding a certain percentile ranking R_i. The resulting quality function is: $Q_i(R_i) = Q_{MAX}$ when $R_i \geq a$, and $\frac{R_i}{a}(Q_{MAX} - Q_{BestEffort}) + Q_{BestEffort}$ otherwise. By increasing its contribution, a producer expects to increase its ranking as compared to other producers, and thus its quality function.

Metric scope The metric is each entity's score, which is globally known. The score can be computed in different ways, but is a function of the entity's contribution. Supply/demand conditions on the network could be reflected in variable rewards for the same contribution, or a punishment applied for refusing to honor a request. Furthermore, scores can be subjected to decay over time.

Analysis

Applying INCA framework allows us to highlight some further research areas for the preceding publication.

- An entity will co-operate until it obtains a ranking that makes its quality function equal to Q_{MAX}: increasing costs would make its utility smaller if it continues to co-operate. As other entities co-operate, their score and thus their ranking increase, lowering the relative ranking of the non co-operating entity. By increasing its co-operation, an entity can then maximize its utility. In cases where other entities are also co-operating, an entity might not be able to increase its ranking substantially, while still incurring costs associated with providing services. A situation would then arise where an entity, even willing to co-operate, is better off by not co-operating even when its quality function is close to $Q_{BestEffort}$. This situation of 'score inflation' could be caused by malicious nodes and is illustrated in Figure 4.7.

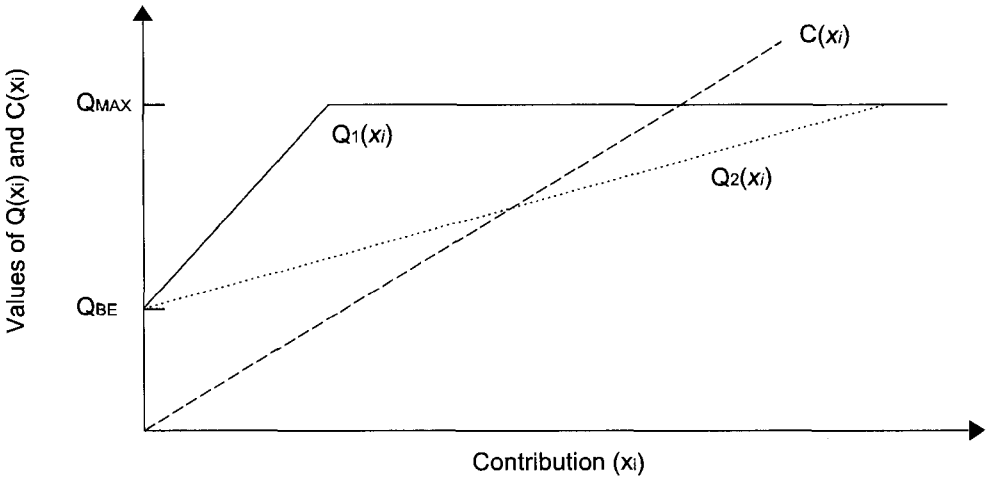

Figure 4.7: Degradation of incentives due to metric inflation

- Since each entity is responsible for maximizing its own utility, the following situations are valid. An entity with high score can reply to an entity with any score - including lower score. Similarly an entity does not have to honor any request, even if it comes from an entity with higher score. The result is that unless entities have a social conscience and give preference to higher scores, higher scores do not mean that a producer will honor requests for service, or that it won't serve another consumer with a lower score instead, undermining the basic premise of the motivation element.

- Since each entity reports its own score, the temptation to misrepresent it in order to gain an advantage are high since tangible benefits are at stake. Additionally, the risk of collusion between malicious entities needs to be taken into account. The authors identify reputation management systems as a possible solution to those problems. All the scores are then weighted by the degree of trust and/or reputation of the reporting and referring entity. Since both the score and trust-worthiness of an entity reflect its degree of co-operation, collecting scores appears then to be redundant.

- In order to compute the utility function, an entity has to compute its ranking within the network. This computation requires $O(\sigma^2)$ samples, where σ is the standard

deviation of the population. This σ is *a priori* not known, and thus the number of samples taken can introduce variable errors into the calculation of the utility function.

4.3.3 Incentives in Distributed Storage

An architecture for fair sharing of resources is presented in [104]. It nominally addresses storage resources, but can be extended to bandwidth as well. An incentive system ensures that participating entities can consume only as much resources as they themselves provide in return to the network. A regime of audits is instituted in order to prevent entities from gaining more than their share of resources.

Motivation Participating entities can take advantage of resources on the network, provided that they make available an equal amount of resources available to other entities. All well-behaved entities will receive the same service, and in contrast to other systems where a malicious (or free-riding) entity will have its quality of service lowered or curtailed, a non co-operating entity will be expelled from the network. The motivation thus comes from the fear of expulsion.

Differentiation Each entity publishes a *usage file*, which contains information about its offered service and currently held contracts with other entities. Consumers simply need to find a producer that offers an amount of storage equal to or greater than their storage requirements: producers thus differentiate themselves by self-description. Producers on the other hand need to perform a simple calculation dependent on consumer's request and usage file. All producers perform the same calculation in order to reject consumers that are over their allowed storage quota: consumer differentiation is thus a group-driven calculation.

Peer selection Peer selection is static for both consumers and producers.

Utility calculation Utility calculation is not explicitly discussed by the authors, but is straight-forward. Consumers are only interested in being able to use network's resources whenever they need to, and that is a function of how much resources they offer in return. Producers on the other hand will only provide resources to well-behaved consumers, and as such are concerned with social welfare.

Metric scope As mentioned previously, entities use information contained in the usage file, which is global in scope.

Analysis

Applying INCA framework allows us to highlight some further research areas for the preceding publication.

- Assuming that identities are not easy to obtain, a malicious peer's identity needs to be propagated throughout the whole network in order to ensure that it cannot enter into a contract with other peers. The main issue however is to ensure that a malicious peer cannot rapidly return with another identity.
- Assume that during a regular audit (where a consumer verifies that its content is truly stored by a given producer), the consumer discovers that the producer is malicious. The producer however passes audits by other consumers. Should the producer be punished (expelled from the network)?
- Following on the preceding point, what happens if a consumer claims - maliciously - that a producer is not storing files it claims to store?
- In order to motivate entities to be truthful, regular and random audits need to be performed. Whereas it is understandable that a consumer would take an interest that its files are properly stored - even at a cost to the consumer, performing random audits is less so. In fact, an entity could free-ride, and let other entities perform random audits. The benefit of performing a random audit accrues to all participants, but the cost is borne only by the auditing parties.
- A consumer, without being malicious, could still obtain more than its fair share of network resources by following this simple scenario: before requesting resources, increase its own offered resources. After allocating its files to producers for storage, decrease its own offered resources. Auditing, both regular and random, would not be able to detect this anomaly.

4.3.4 Incentives in Content Distribution

In order to improve the performance of data-sharing networks, [40] presents a protocol for creation of self-organizing topologies based on peers' assessment of which direct connections are most likely to result in satisfactory file download. This results in formation of clusters of well-behaved entities, which have a broader view of the network than non co-operative entities.

Motivation It can be argued that motivation in this case is a combination of fear and greed (carrot and stick approach). Knowledge of entities' actions (choice of co-operation or non co-operation) is accumulated throughout the network. After a sufficiently long time, each entity can be judged on the difference between the sum of co-operation instances and the sum of non-co-operation instances.

Differentiation Entities select *good* peers to connect to based on history of past interactions. Differentiation therefore is based on the result of a calculation by an entity.

Peer selection Peer selection is dynamic. Entities constantly update their connections with peers that maximize their utility function.

Utility calculation All entities are interested in increasing only their own utility. The network is defines as an undirected graph with $G = (P, E)$ where P is the set of entities and E is the set of connections such that $c(i,j) = 1$ when there is a connection, $c(i,j) = 0$ when there is no connection, and $c(i,i) = 0$ for $i, j \in P$. The objective function is defined as $Q_i = \sum_{j=1}^{v} c(i,j)s_{ij}$ subject to $\sum_{j=0} c(i,j) \leq \tau \; \forall i$. The number of satisfactory interactions s_{ij} is defined as $s_{ij} = satisfactory(i,j) - unsatisfactory(i,j)$.

Metric scope Metric is a combination of local trust, connection trust and void downloads. Entities assess other entities' co-operation from their own experience, but do not share this information with other entities; scope of the metric is thus local.

Analysis

Applying INCA framework allows us to highlight some further research areas for the preceding publication.

- In order to establish connection with well behaved peers, an entity must first learn about the entities it interacts with. This is a function of the size of the network, cycle time - the time it takes to make a decision on the success of an interaction, e.g. downloading a movie is slower than downloading a song, availability of entities - time they spent connected to the network, and the number of connections each peer can have. All of these have an effect on convergence to a state that is stable and/or acceptable to a peer.

- Assuming a stable state is found, entities that join the network will have no history of previous interaction and thus it will be difficult for them to establish links with well behaved peers.

4.4 Comparable Work

The body of literature describing frameworks for incentive mechanisms encompasses both quantitative and qualitative studies. In the latter case, [108] considers distributed systems composed of autonomous entities co-operating to achieve their goals. Authors observe that incentive schemes play a central role, and propose a classification of incentive patterns. Matching these incentive patterns with specific application environment allows for a more systematic approach to the design of incentive schemes.

Quantitative studies typically analyze incentive mechanisms in the context of game theory [122]. The performance of three incentive schemes, token-exchange, peer-approved and service-quality, is analyzed in [114] by defining and using a model based on Multi-Person Prisoner's Dilemma. A generalized form of Evolutionary Prisoner's Dilemma model is used to study cooperation in peer-to-peer networks [86]. Authors conclude that even with unfettered and decentralized creation of identities, adaptable incentive mechanisms can produce systems that converge to full cooperation. Furthermore, scalability and security of networks is addressed in the context of private and shared history of entities' actions. A formal game theoretic model for systems relying on centralized servers is presented in [60], and provides an analysis of equilibrium of user strategies for various payment schemes. With the view to assess and evaluate incentive systems in peer-to-peer networks, [14] presents a model of utilities obtained by entities, and various schemes to address the lack of incentives. Analytical expressions are provided for cases where there is a lack of complete payoff information and an inability to individually compensate entities. Authors' main findings relate to the size of the network and variability of entities' utility functions. Another study presents a discussion of Nash equilibrium for a non co-operative game between strategic players [32]. Assuming that the probability of obtaining a service from a producer depends on the contribution of a consumer (its production), this study addresses cases when the benefit to a peer from other entities' contribution is common to all, and when the benefit is specific to each peer.

In networks with autonomous and transient entities, the supply and demand of services is likely to fluctuate. It would be desirable to design incentive mechanisms to take into account the dynamics of peer-to-peer networks, offering higher compensation for scarce services, and lower for abundant ones. More generally, for each entity having a set of strategies and preferred outcomes, the challenge is to design a utility function that at equilibrium yields a certain desirable network property [51]. This, in essence, is the domain of mechanism design. Work presented in [149] designs a utility function, and

provides a solution using control theoretic approach.

4.5 Summary

This chapter presents INCA, a qualitative reference framework for incentive systems. All such systems can be de-constructed into five elements, incentive, differentiation, peer selection, utility calculation and metric scope. Functional and effective incentive mechanism will have all of those elements, except possibly the peer selection element. Each element presents some design choices that will define characteristics of an incentive mechanism.

Chapter 5

Leveraging Inter-Service Redundancy

5.1 Applicability of FSMs to Represent Services

Finite State Machines (FSMs) are well-known formal concept, which finds wide application in software specification, specification and verification of protocols, sequential logic circuits design or control systems (software and hardware) [29, 81, 68, 62].

In the context of electronic commerce, work presented in [6] models the interaction of an application whose state is described by a relational database (business model) as a restricted class of transducers (semi-positive outputs and cumulative state), which the authors argue, capture wide range of practically significant business models.

The Business Process Execution Language (BPEL) for Web Services [106] builds on top of WSDL [2] to allow the exchange of message sequences between entities. Work presented in [50] formalizes key functional attributes of BPEL based on the asynchronous computation model of distributed abstract state machines (ASM) [62], while [28] uses distributed ASMs to analyze business process workflow patterns. An algorithm to derive a finite state machine from a given abstract state machine specification is given in [61].

Usage of FSMs to formally describe services (as *conversations* a service can have) has been used in [20], which was presented in section 2.2.3. The communication between components of a composite service represented as FSMs is considered in [31], where authors show that composite services may no longer be a FSM in presence of unexpected behavior. Noting that only regular activities (interaction with the service) are considered in [20, 31], and that more complex and non-regular activity sequences are possible,

infinite-state automata are considered in [42].

Having presented related research work that uses FSM for representations of services and their limitations, we note that the purpose of the present thesis is not to find a new or best formal (see section 2.2.1 for other methods, for example) representation of services, but to argue that inter-service redundancy can be exploited to improve the robustness of applications that depend on those services.

5.2 Representation of Services

A consumer C and producer P interact through message passing. We assume that all messages are parametrized on the type of message and are of the form $c_i(data)$ and $p_j(data)$ where c_i are messages sent by the consumer and received by the producer and p_j are messages sent by the producer and received by the consumer. Parametrization on the type of message [31] allows us to decouple the data that is passed to the producer from the message itself. For each received non-null message, the producer replies with exactly one message ($c_i \rightarrow p_j$), where p_j can be a null-message ($c_i \rightarrow \epsilon$). For the same input sequence, a producer will send the same output sequence (otherwise service would be random). The result of this interaction is some service that the consumer receives from the producer:

Definition 5.2.1. *(Service) Given a finite sequence s_c of messages c drawn from some finite alphabet $\mathbb{C}$ and a finite sequence s_p of messages p drawn from some finite alphabet $\mathbb{P}$, a service is a tuple (s_c, s_p) such that $|s_c| = |s_p|$*

Example 1: Given the messages $c_1(data)$ = `calculateSquareRoot`(data), $c_2(data)$ = `calculateAverage`(data) and $p_1(data)$ = `result(data)`, then two services are $s_1 = (c_1(9), p_1(3))$ and $s_2 = (c_2(49, 51), p_1(50))$.

Example 2: In the case of an on-line purchase of books, a service can be represented as $s_3 = (c_{10}c_{11}c_{12}, p_1p_2p_3)$ or $s_4 = (c_{10}c_{11}c_{11}c_{11}c_{12}, p_1p_2p_2p_2p_3)$

where c_{10} = `searchListContaining`("finite", "state", "machines")
c_{11} = `addToCart`(BookID)
c_{12} = `checkOut`(shippingInfo, creditCardDetails)
p_1 = `result`(bookList)
p_2 = `cartContents`(list)

$p_3 =$ `transactionResult`("true")

The choice of service definition in definition 5.2.1 constrains how the interaction is described, for instance, whereas the "pull" case is readily handled, the "push" case (the producer sends a message on a null-message) would need to be handled differently. If, for instance, a consumer is interested in the membership change of a group (join or leave), a service tracking this change can either be polled (pull) or send membership change information to all that indicated their interest in that information as it happened (push). Presumably, this information would only be sent if the consumer were in a state to receive it, as evidenced for example by periodic "keep alive" messages. The push case of such a service could then be represented as $(c_k c_k \cdots c_k, \epsilon\epsilon \cdots p_l)$ where $c_k =$ `heartbeat`() and $p_l =$ `groupMembership`().

As noted in the last example, a provider can render various services (s_3 and s_4) that achieve the same goal from the point of view of the consumer: purchasing books. The producer can specify those services compactly as a finite-state machine:

Definition 5.2.2. *(Service Specification) A service specification is a finite-state transducer* $S_k = (\Sigma, \Gamma, Q_k, q_0, \delta_k, \omega_k)$

where $\Sigma =$ *is a finite input alphabet, or the set of all messages that can be received by the producer*

$\Gamma =$ *is a finite output alphabet, or the set of all messages that can be send by the producer*

$Q_k =$ *represents a finite set of states*

$q_0 =$ *represents the initial state of the machine where* $q_0 \in Q_k$

$\delta =$ *is a transition function where* $Q \times \Sigma \rightarrow Q$

$\omega =$ *is a output function* $Q \times \Sigma \rightarrow \Gamma \cup \epsilon$

Note that we omit the final (or accepting) states from the definition 5.2.2 and assume that all $q \in Q_k$ are final, since in different contexts, different final states might be specified. Alternatively, instead of the transition and output function, we can use a set of relations $E \subseteq Q \times \Sigma \times \Gamma \times Q$.

Definitions 5.2.1 and 5.2.2 are similar to the approach taken elsewhere [20, 31], however these are based on finite-state automata and therefore use different classes of algorithms.

There are also alternative service representation to the one proposed here, which for example, would allow the producer to send an extra message in addition to the output

Name	Var	Comments
Redundancy Graph	G	vertices V and edges E
Redundancy Subgraph	G_s	vertices V_s and edges E_s
Maximum Redundancy Subgraph	G_s^{max}	vertices V_s and edges E_s^{max}
Target FST	T	service specification to be created
Candidate FST	C	service specification that is available
Functional Separation FST	T_α	machine relating T_β and T_γ
Redundancy Reuse FST	T_β	machine encoding G_s^{max}
State Support FST	T_γ	machine encoding states and transitions not contained in G_s^{max}

Table 5.1: Summary of notation used in the subsequent algorithms

message caused by some specific input message sequence ($c_i \rightarrow p_j p_k p_l$), or to send an output message even if no input was received ($\epsilon \rightarrow p_j$). Such representations are more flexible when modeling a system, but involve much more complex - sometimes impractical - methods and algorithms. Note that those two examples correspond to subsequential (or p-subsequential) and nondeterministic machines respectively.

The relationship between a service and a service specification is given by:

Definition 5.2.3. *(Service Conformance) A service s conforms to a service specification S if and only if the service s is a regular relation computed by S.*

With those definitions, we can now proceed to discuss the similarity of various service specifications.

5.3 Redundancy Calculation

The following discussion uses the notation summarized in table 5.1. We start by creating a graph $G = (V, E)$ from the target and candidate specification where vertices are given by $Q_T \times Q_C$ relation and edges are defined as a subset of $E_T \times E_C$. Given $(q, \sigma, \gamma, r) \in E_T$ and $(q', \sigma, \gamma, r') \in E_C$, we define $E_{qr} = \{((q, q'), \sigma, \gamma, (r, r')) \mid \exists (q, \sigma, \gamma, r) \in E_T \wedge \exists (q', \sigma, \gamma, r') \in E_C, q \neq r, q' \neq r'\}$ and $E_{qq} = \{((q, q'), \sigma, \gamma, (q, q')) \mid \exists (q, \sigma, \gamma, q) \in E_T \wedge \exists (q', \sigma, \gamma, q') \in E_C\}$. Then,

Definition 5.3.1. *(Redundancy Graph) The redundancy graph of a service specification*

$T = (\Sigma_T, \Gamma_T, Q_T, q_T, E_T)$ *with respect to a service specification* $C = (\Sigma_C, \Gamma_C, Q_C, q_C, E_C)$ *is a graph* $G = (V, E)$ *where* $V = Q_T \times Q_C$ *and* $E = E_{qr} \cup E_{qq}$.

Definition 5.3.2. *(Redundancy Subgraph) The redundancy subgraph of* $G = (V, E)$ *is a graph* $G_s = (V_s, E_s)$ *where* V_s *is any binary relation on* $Q_T \times Q_C$ *with the following properties:*

i) if $(t, c) \in V_s$ *and* $(t, c') \in V_s$, *then* $c = c'$

ii) if $(t, c) \in V_s$ *and* $(t', c) \in V_s$, *then* $t = t'$

iii) if $|Q_T| \leq |Q_C|$ *then* $\forall t \in Q_T, \exists c \in Q_C$ *such that* $(t, c) \in V_s$

iv) if $|Q_T| > |Q_C|$ *then* $\forall c \in Q_C, \exists t \in Q_T$ *such that* $(t, c) \in V_s$

Definition 5.3.2 basically states that V_s defines $|Q_T|$ or $|Q_C|$ one-to-one relations between states of Q_T and Q_C. We can now define the terms *redundancy* and *maximum redundancy*:

Definition 5.3.3. *(Redundancy) The redundancy of a service specification* T *with respect to a service specification* C *is a tuple* $(G_s, \mathbb{I})$ *where* $G_s = (V_s, E_s)$ *is any redundancy subgraph of graph* G *such that* $|E_s| > 0$ *and* $\mathbb{I} = V_s \cap \{q_T \times q_C\}$

Definition 5.3.4. *(Maximum Redundancy) The maximum redundancy of a service specification* T *with respect to a service specification* C *is a tuple* $(G_s^{max}, \mathbb{I})$ *where* $G_s^{max} = (V_s, E_s^{max})$ *is a subgraph of* G *such that* $\forall E_s, |E_s^{max}| \geq |E_s|$ *and* $\mathbb{I} = V_s \cap \{q_T \times q_C\}$

Note that a brute-force approach to find G_s^{max} requires a comparison of $|Q_T| \times |Q_C| \times (|Q_T| - 1) \times (|Q_C| - 1) \times (|Q_T| - 2) \times (|Q_C| - 2) \ldots \times (|Q_T| - n) \times (|Q_C| - n)$, where $n = min(Q_T, Q_C)$ or $\frac{|Q_T|!|Q_C|!}{|Q_T - Q_C|!}$ possibilities. Such an approach quickly becomes computationally intensive, since for even a small number of states, for example $Q_T = Q_C = 7$ there are over 25×10^6 possibilities.

To find a local maximum, a heuristic based on finding successive nodes with the greatest number of incoming and outgoing transitions connected to previously selected nodes is used. The number of comparisons necessary is $|Q_T| \times |Q_C| + (|Q_T| - 1) \times (|Q_C| - 1) + (|Q_T| - 2) \times (|Q_C| - 2) \ldots + (|Q_T| - n) \times (|Q_C| - n)$, or $\sum_{n=0}^{N} (|Q_T| - n)(|Q_C| - n)$ where $N = |Q_C| - 1$ and assuming $|Q_C| \leq |Q_T|$. Note that the number of comparisons is less that $|Q_T| \times |Q_C| \times |Q_C|$ since $\sum_{n=0}^{N} n(n - |Q_T| - |Q_C|) \leq 0$.

The algorithm to find the maximum redundancy of a target with respect to a candidate is given in algorithm 1 and uses three functions, MARK, MATCH and MAXG listed below.

Algorithm 1: Redundancy

input : FST $T = (\Sigma_T, \Gamma_T, Q_T, q_T, E_T)$, FST $C = (\Sigma_C, \Gamma_C, Q_C, q_C, E_C)$

output: Graph $G_s^{max} = (V_s, E_s^{max})$, function `H()` and Boolean I

1 $(e^o_{Q_T \times Q_C}, e^i_{Q_T \times Q_C}) \leftarrow$ `Mark(`T, C`)`

2 `H` $\leftarrow$ `Match(`$e^o_{Q_T \times Q_C}, e^i_{Q_T \times Q_C}$`)`

3 $G_s^{max} = (V_s, E_s^{max}) \leftarrow$ `MaxG(H())`

4 $I \leftarrow (q_T \in V_s) \wedge (q_C =$ `H(`q_T`))`

The purpose of the first function is to find all the transitions common to both service specifications. A transition is common to both specifications just in case it has the same input and output labels and in each specification, leads either back to the originating state (self-transition), or to a different state. The number of possible comparisons is $|Q_T| \times |Q_C| \times |\Sigma_T \cap \Sigma_C|$ or $|Q_T| \times |Q_C| \times |\Sigma_C|^2$ if we assume that $\Sigma_T = \Sigma_C$. The size of the output alphabet Γ does not have an impact since there can only be one transition with a particular input label (the underlying finite-state transducer is deterministic).

The purpose of the second function is to find a "partial" one-to-one correspondence between states of both specifications (in the sense that the number of states may differ). As previously discussed, assuming $|Q_C| \leq |Q_T|$ the number of necessary comparisons is less than $|Q_T| \times |Q_C| \times |Q_C|$.

Finally, the third function filters out all states and transitions of the target service specification that do not have a counterpart in the candidate specification. Again, assuming $|Q_C| \leq |Q_T|$, the maximum number of comparisons to make is $|Q_C| \times |\Sigma_C|$.

For illustration purposes, the target and candidate machines shown in figures 5.1 and 5.2 respectively will be used as an example. We now want to create a machine that contains the target specification using the maximum redundancy just found.

5.4 Target Decomposition

The maximum redundancy (states and transitions) suggests that there is a distinction between the sequences that can be handled by the candidate machine and those that cannot. We therefore look for a decomposition of the target machine into distinct parts; the first which encodes the relationship between those two parts, the second which encodes sequences handled by the candidate machine and contains all states and transitions specified by G_s^{max}, and finally, the third which encodes the remaining sequences. Those

Function `Mark`

input : FST $T = (\Sigma_T, \Gamma_T, Q_T, q_T, E_T)$, FST $C = (\Sigma_C, \Gamma_C, Q_C, q_C, E_C)$

output: Set of transitions $e^o_{u\cdot v}$ and $e^i_{u\cdot v}$ for each $q_{u\cdot v} \in Q_T \times Q_C$

forall $u \in Q_T$ **do**
  forall $v \in Q_C$ **do**
    forall $(u, \sigma_1, \gamma_1, r) \in E_T$ **do**
      forall $(v, \sigma_2, \gamma_2, s) \in E_C$ **do**
        if $(\sigma_1 = \sigma_2 \wedge \gamma_1 = \gamma_2)$ **then**
          if $(u \neq r \wedge v \neq s) \vee (u = r \wedge v = s)$ **then**
            $e^o_{u\cdot v} \xleftarrow{+} (u \cdot v, \sigma_1, \gamma_1, r \cdot s)$
            $e^i_{r\cdot s} \xleftarrow{+} (u \cdot v, \sigma_1, \gamma_1, r \cdot s)$
      end
    end
  end
end
return $e^o_{Q_T \times Q_C}$, $e^i_{Q_T \times Q_C}$

three machines are referred to as T_α, T_β and T_γ respectively. The desired result of the composition $T_\alpha \circ T_\beta \circ T_\gamma$ is a machine that contains the target machine T, and is shown in figure 5.7.

In the process of decomposing the target machine, we need to create some special identifiers, by adding a reference to the state [119], that are not contained in either the input or output alphabets and which can be used in addition to the original transition labels. These identifiers will not be used as input to, or output from the candidate machine, and must preserve the structure of the finite state machine with respect to its sequentiality. A straight-forward manner to generate those new symbols is string homomorphism which replaces a symbol with a particular string, the method `concatenate`(σ,q) for instance creates a new label by concatenating strings used to represent a particular symbol and state.

5.4.1 Functionality Separation

The functional separation machine (T_α) contains only two states $Q_\alpha = \{r, s\}$, representing the fact that the work is performed by either the T_β or T_γ machine and its starting state depends on whether the initial state of the target belongs to V_s and the transitions

Function `Match`

input : Set of transitions $e^o_{u\cdot v}$ and $e^i_{u\cdot v}$ for each $q_{u\cdot v} \in Q_T \times Q_C$

output: Correspondence between states Q_T and Q_C given by `H()`

1 $\mathsf{row} \leftarrow Q_T$

2 $\mathsf{col} \leftarrow Q_C$

3 $\mathtt{H}(u,v) \xleftarrow{+} \underset{q_{u\cdot v} \in Q_T \times Q_C}{\operatorname{argmax}} \left(|e^o_{u\cdot v}| + |e^i_{u\cdot v}| - |e^o_{u\cdot v} \cap e^i_{u\cdot v}|\right)$

4 **while** $(|\mathsf{row}| > 1 \wedge |\mathsf{col}| > 1)$ **do**

5 $\quad \hat{Q} \leftarrow (\mathsf{row} \xleftarrow{-} u) \times (\mathsf{col} \xleftarrow{-} v)$

6 $\quad \hat{S} \leftarrow \{q_{r\cdot c} \mid \exists \mathsf{key} \in \mathtt{H}(), (q_{r\cdot c}, \sigma, \gamma, q_{\mathsf{key}\cdot\mathtt{H}(\mathsf{key})}) \in e^o_{r\cdot c}, q_{r\cdot c} \in \hat{Q}\}$

7 $\quad \hat{S} \xleftarrow{+} \{q_{r\cdot c} \mid \exists \mathsf{key} \in \mathtt{H}(), (q_{\mathsf{key}\cdot\mathtt{H}(\mathsf{key})}, \sigma, \gamma, q_{r\cdot c}) \in e^i_{r\cdot c}, q_{r\cdot c} \in \hat{Q}\}$

8 $\quad \mathtt{H}(u,v) \xleftarrow{+} \underset{q_{u\cdot v} \in \hat{S}}{\operatorname{argmax}} \left(|e^o_{u\cdot v}| + |e^i_{u\cdot v}| - |e^o_{u\cdot v} \cap e^i_{u\cdot v}|\right)$

9 **end**

10 **return** `H()`

Function `MaxG`

input : Correspondence between states Q_T and Q_C given by `H()`

output: Graph given by G^{max}_s

1 **forall** $\mathsf{key} \in \mathtt{H}()$ **do**

2 $\quad V_s \xleftarrow{+} \mathsf{key}$

3 $\quad E^{max}_s \xleftarrow{+} \{(\mathsf{key}, \sigma, \gamma, r) \mid \forall r \in \mathtt{H}(), (q_{\mathsf{key}\cdot\mathtt{H}(\mathsf{key})}, \sigma, \gamma, q_{r\cdot\mathtt{H}(r)}) \in e^o_{\mathsf{key}\cdot\mathtt{H}(\mathsf{key})}\}$

4 **end**

5 **return** $G^{max}_s = (V_s, E^{max}_s)$

are defined by algorithm 5. The maximum number of comparisons is given by $|Q_T| \times |\Sigma_C|$, assuming $\Sigma_T = \Sigma_C$. The global machine is shown in figure 5.4.

5.4.2 Redundancy Reuse

The redundancy reuse machine (T_β) encodes the information about how the candidate machine should be accessed in order to use only that part which produces the maximal redundancy. A "foreign" state is created which represents any state that is not contained in G^{max}_s. There will be two other types of transitions in addition to those specified in G^{max}_s. The first type encodes information about transitions where both vertices are

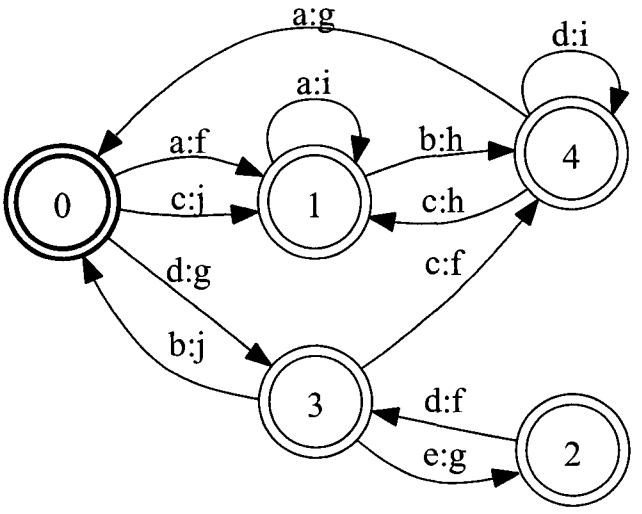

Figure 5.1: Target service specification

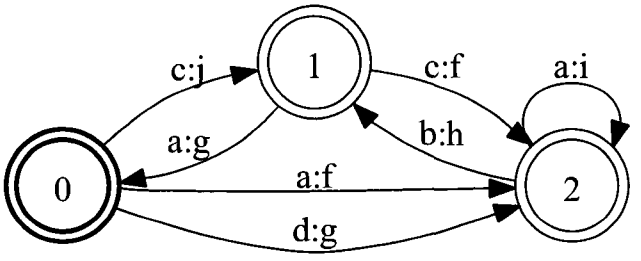

Figure 5.2: Candidate service specification

in the foreign state, and the second where one vertex is in the foreign state. The T_β machine is shown in figure 5.5, and is produced by algorithm 6. The maximum number of comparisons is given by $|Q_T| \times |\Sigma_C|$, assuming $\Sigma_T = \Sigma_C$.

5.4.3 Service State Support

The state support machine (T_γ) encodes the information about the states and transitions that are not contained in G_s^{max}. The procedure is similar as for the T_β machine except that the "foreign" state represents now all states contained in G_s^{max} and instead of using input labels we use output labels. The T_γ machine is shown in figure 5.6, and is produced by algorithm 7. The maximum number of comparisons is given by $|Q_T| \times |\Sigma_C|$, assuming

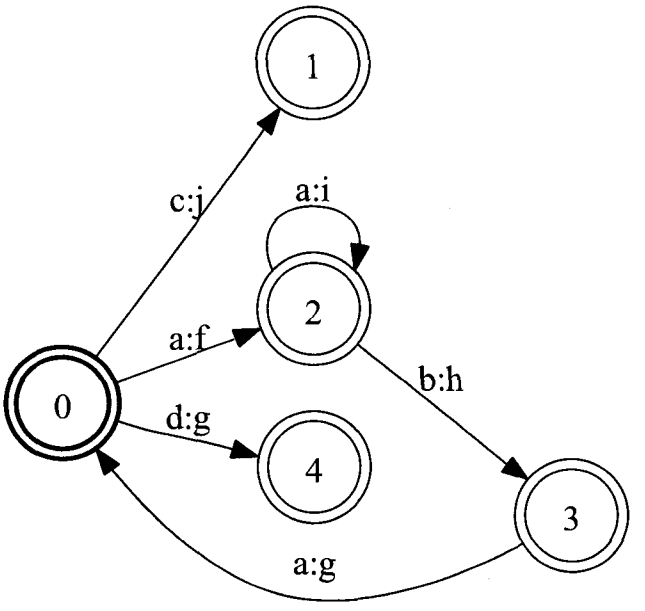

Figure 5.3: Intersection of target and candidate machines

$\Sigma_T = \Sigma_C$.

5.5 Implementation Notes

T_β machine encodes the information about how the candidate machine should be accessed. This information must be interpreted in some way in order to realize an implementation. Any implementation would take advantage of the fact that regular relations are closed under concatenation, union and compositions operations (see section A.1.7). We first discuss three basic cases and then the two remaining complex cases which rely on the basic cases for implementation.

5.5.1 Transition to Start State

Whenever we have a transition σ_m/γ_n from a foreign state F to a state of T_β, it implies that a transition should be made into the candidate machine C. This particular case applies when $q_i \in T_\beta$ corresponds to the start state of machine C, in effect starting a new instance of the candidate machine. Those transitions can be expressed as $F \xrightarrow{\sigma_m/\gamma_n} (q_i \mid q_i \in Q_\beta, q_i = q_C)$.

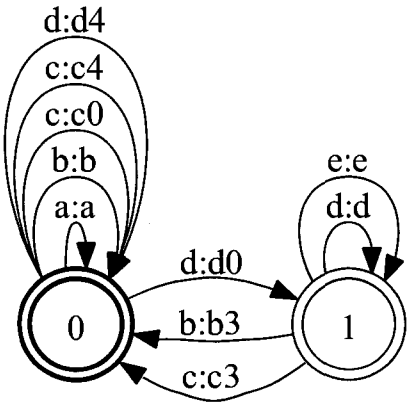

Figure 5.4: T_α machine, where states $\{0, 1\}$ represent $\{r, s\}$ respectively

If a finite-state transducer τ_1 computes a relation $|\tau_1|(\sigma_m) = \{\gamma_n\}$, then the concatenation $\tau_1 \cdot C$ results in a finite-state transducer $(\Sigma'_C, \Gamma'_C, Q'_C, F, E'_C)$ where $\Sigma'_C = \Sigma_C \cup \sigma_m$, $\Gamma'_C = \Gamma_C \cup \gamma_n$, $Q'_C = Q_C \cup F$ and $E'_C = E_C \cup (F, \sigma_m, \gamma_n, q_C)$.

5.5.2 Transition to Foreign State

Similarly to the previous case, whenever we have a transition σ_m/γ_n from some state $q_i \in T_\beta$ to the foreign state F, it implies that a transition should be made out of the candidate machine C, in effect leaving the current instance of the candidate machine. Those transitions can be expressed as $(q_i \mid q_i \in Q_\beta) \xrightarrow{\sigma_m/\gamma_n} F$.

If a finite-state transducer τ_2 computes a relation $|\tau_2|(\sigma_m) = \{\gamma_n\}$, then the concatenation of $C \cdot \tau_2$ results in a finite-state transducer $(\Sigma'_C, \Gamma'_C, Q'_C, q_C, E'_C)$ where $\Sigma'_C = \Sigma_C \cup \sigma_m$, $\Gamma'_C = \Gamma_C \cup \gamma_n$, $Q'_C = Q_C \cup F$ and $E'_C = E_C \cup (q_i, \sigma_m, \gamma_n, F)$.

5.5.3 Direct Connection Between States

The case where a transition σ_m/γ_n must be made from one state of T_β to another when there exists an extended transition function such that $\hat{\delta}(q_i, s) = q_j$ (equation A.1) implies that we remain in the current instance of the candidate machine, but that both the input and output labels of the transition do not belong to the alphabet of the candidate machine. Those transitions can be expressed as $q_i \xrightarrow{\sigma_m/\gamma_n} (q_j \mid q_i, q_j \in Q_\beta, \exists\ \hat{\delta}(q_i, s) = q_j, s \in \Sigma^*)$.

If two finite-state transducers compute the following relations, $|\tau_{3a}|(\sigma_m) = \{s\}$ and

Algorithm 5: createAlpha

input : FST $T = (\Sigma_T, \Gamma_T, Q_T, q_T, E_T)$, graph $G_s^{max} = (V_s, E_s^{max})$ and Boolean I
output: FST $T_\alpha = (\Sigma_\alpha, \Gamma_\alpha, Q_\alpha, q_\alpha, E_\alpha)$

$Q_\alpha \leftarrow (r, s)$
if (I) **then** $q_\alpha \leftarrow r$ **else** $q_\alpha \leftarrow s$
forall $q \in Q_T$ **do**
  forall $(q, \sigma, \gamma, q') \in E_T$ **do**
    symbol ←concatenate(σ,q)
    if $(q \in V_s \wedge q' \in V_s)$ **then**
      if $(q, \sigma, \gamma, q') \in E_s^{max}$ **then** $(\Sigma_\alpha, \Gamma\alpha, E_\alpha) \xleftarrow{+} (\sigma, \sigma, (r, \sigma, \sigma, r))$
      else $(\Sigma_\alpha, \Gamma\alpha, E_\alpha) \xleftarrow{+} (\sigma, \text{symbol}, (r, \sigma, \text{symbol}, r))$
    if $(q \notin V_s \wedge q' \notin V_s)$ **then**
      $(\Sigma_\alpha, \Gamma\alpha, E_\alpha) \xleftarrow{+} (\sigma, \sigma, (s, \sigma, \sigma, s))$
    if $(q \in V_s \wedge q' \notin V_s)$ **then**
      $(\Sigma_\alpha, \Gamma\alpha, E_\alpha) \xleftarrow{+} (\sigma, \text{symbol}, (r, \sigma, \text{symbol}, s))$
    if $(q \notin V_s \wedge q' \in V_s)$ **then**
      $(\Sigma_\alpha, \Gamma\alpha, E_\alpha) \xleftarrow{+} (\sigma, \text{symbol}, (s, \sigma, \text{symbol}, r))$
  end
end
return T_α

$|\tau_{3b}|(w) = \{\gamma_n\}$, where $w \in \Gamma^*$ is given by the extended output function $\hat{\omega}(q_i, s) = w$ (equation A.2), then the composition of $\tau_{3a} \circ C \circ \tau_{3b}$ results in a finite-state transducer which computes the σ_m/γ_n relation.

5.5.4 Transition to Arbitrary State

Whenever we have a transition σ_m/γ_n from a foreign state to a state of $q_i \in T_\beta$ when q_i is not the start state of machine C, it implies that a new instance of the candidate machine C should be created and its state moved to the desired state before accepting any inputs. Those transitions can be expressed as $F \xrightarrow{\sigma_m/\gamma_n} (q_i \mid q_i \in Q_\beta, q_i \neq q_C)$.

This case can be handled by noticing that we can first create a transition $F \xrightarrow{\sigma_{m'}/\gamma_{n'}} q_C$ as in case 5.5.1, and then obtain a direct transition $F \xrightarrow{\sigma_m/\gamma_n} q_i$ as in case 5.5.3.

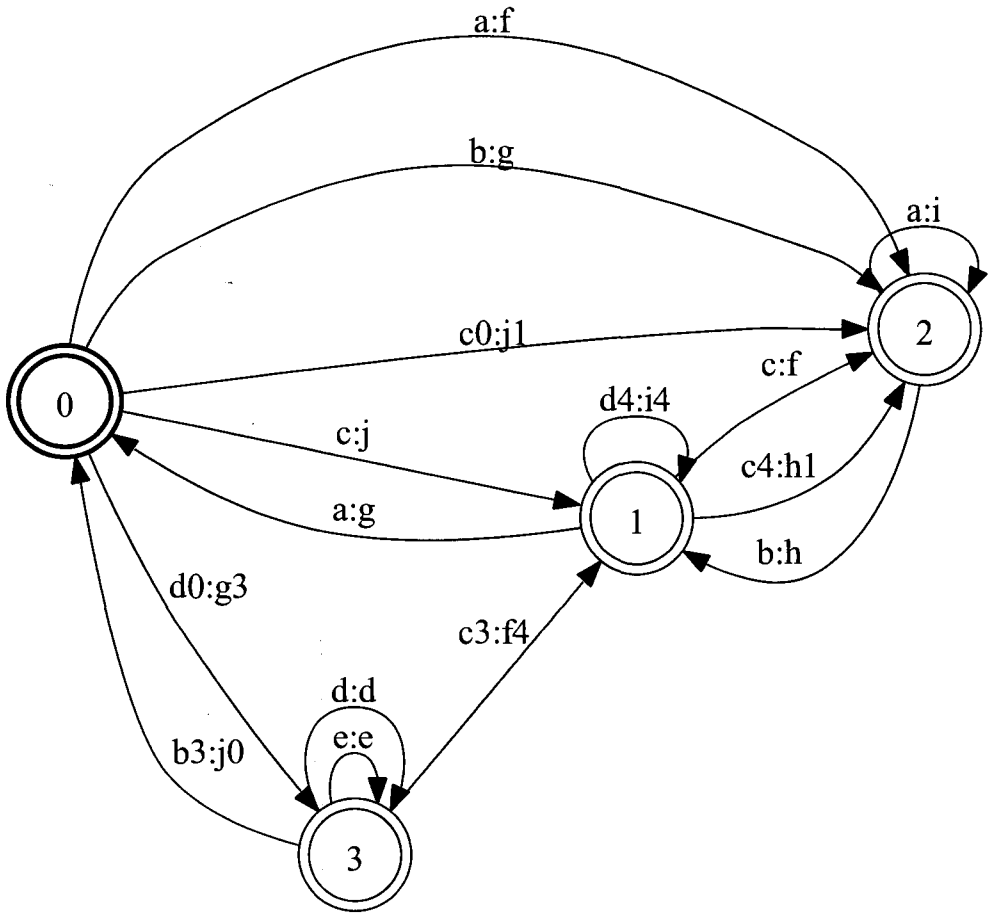

Figure 5.5: T_β machine

5.5.5 Indirect Connection Between States

The case when a transition σ_m/γ_n must be made from $q_i \in T_\beta$ to $q_j \in T_\beta$ but no extended transition function $\hat{\delta}(q_i, s) = q_j$ (equation A.1) exists, implies leaving the current instance of the candidate machine and moving to an arbitrary state of a new instance of a candidate machine. Those transitions can be expressed as $q_i \xrightarrow{s/w} (q_k \mid q_i, q_j, q_k \in Q_\beta, q_k \neq q_j, \forall\, \hat{\delta}(q_i, s) = q_k, s \in \Sigma^*)$.

This case can be handled by noticing that we can first create a transition $q_i \xrightarrow{\sigma_{m'}/\gamma_{n'}} F$ as in case 5.5.2 and a transition $F \xrightarrow{\sigma_{m''}/\gamma_{n''}} q_j$ as in case 5.5.4 before obtaining a direct transition $q_i \xrightarrow{\sigma_m/\gamma_n} q_j$ as in case 5.5.3.

Algorithm 6: createBeta

input : FST $T = (\Sigma_T, \Gamma_T, Q_T, q_T, E_T)$, FST $C = (\Sigma_C, \Gamma_C, Q_C, q_C, E_C)$, graph $G_s^{max} = (V_s, E_s^{max})$, function $\texttt{index}(Q_T) \rightarrow Q_C$ and Boolean I

output: FST $T_\beta = (\Sigma_\beta, \Gamma_\beta, Q_\beta, q_\beta, E_\beta)$

1 $T_\beta \leftarrow C$
2 $Q_\beta \xleftarrow{+}$ foreign
3 **if** (I) **then** $q_\beta \leftarrow q_C$ **else** $q_\beta \leftarrow$ foreign
4 **forall** $q \in Q_T$ **do**
5 **forall** $(q, \sigma, \gamma, q') \in E_T$ **do**
6 L $\leftarrow$`concatenate(`σ`,`q`)`
7 R $\leftarrow$`concatenate(`γ`,`q'`)`
8 **if** $(q \in V_s \wedge q' \in V_s)$ **then**
9 **if** $(q, \sigma, \gamma, q') \notin E_s^{max}$ **then**
10 $(\Sigma_\beta, \Gamma_\beta, E_\beta) \xleftarrow{+}$ (L, R, (`index(`q`)`, L, R, `index(`q'`)`))
11 **if** $(q \notin V_s \wedge q' \notin V_s)$ **then**
12 $(\Sigma_\beta, \Gamma_\beta, E_\beta) \xleftarrow{+} (\sigma, \sigma,$ (foreign, $\sigma, \sigma,$ foreign))
13 **if** $(q \in V_s \wedge q' \notin V_s)$ **then**
14 $(\Sigma_\beta, \Gamma_\beta, E_\beta) \xleftarrow{+}$ (L, R, (`index(`q`)`, L, R, foreign))
15 **if** $(q \notin V_s \wedge q' \in V_s)$ **then**
16 $(\Sigma_\beta, \Gamma_\beta, E_\beta) \xleftarrow{+}$ (L, R, (foreign, L, R, `index(`q'`)`))
17 **end**
18 **end**
19 **return** T_β

5.6 Summary

This chapter presents the formalization of the concept of redundancy based on the finite-state machine notation, and an algorithm to obtain this redundancy from any pair of services, as well as an algorithm to express one service in terms of another using the redundant functionality between the two services. The second algorithm works by decomposing the target service into three submachines such that the entire functionality of the original target machine can be recreated through application of the composition operation to the submachines. One of the submachines is a modified candidate machine while the remaining submachines represent the locally implemented functionality and

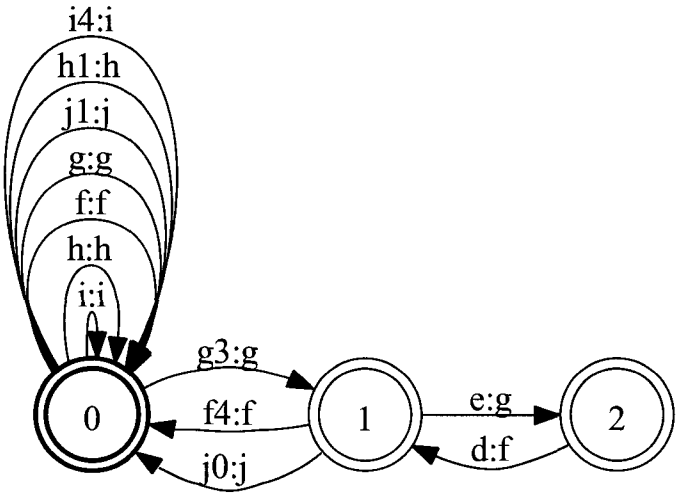

Figure 5.6: T_γ machine

control.

Although the performance of the proposed method is polynomial with respect to the service size, the additional cost is the implementation of the non-redundant functionality and related control locally. Additionally, the proposed method is obviously dependent on the expressive power of the regular relations (transducers), and future work includes investigations into the applicability of other formal ways of representing services.

Algorithm 7: createGamma

input : FST $T = (\Sigma_T, \Gamma_T, Q_T, q_T, E_T)$, graph $G_s^{max} = (V_s, E_s^{max})$ and Boolean I
output: FST $T_\gamma = (\Sigma_\gamma, \Gamma_\gamma, Q_\gamma, q_\gamma, E_\gamma)$

1 $Q_\gamma \leftarrow$foreign
2 **if** (I) **then** $q_\gamma \leftarrow$ foreign **else** $q_\gamma \leftarrow q_T$
3 **forall** $q \in Q_T$ **do**
4 **forall** $(q, \sigma, \gamma, q') \in E_T$ **do**
5 symbol $\leftarrow$`concatenate(`γ`,`q`')`
6 **if** $(q \in V_s \wedge q' \in V_s)$ **then**
7 **if** $(q, \sigma, \gamma, q') \in E_s^{max}$ **then** $(\Sigma_\gamma, \Gamma_\gamma, E_\gamma) \xleftarrow{+} (\gamma, \gamma, (\text{foreign}, \gamma, \gamma, \text{foreign}))$
8 **else** $(\Sigma_\gamma, \Gamma_\gamma, E_\gamma) \xleftarrow{+} (\text{symbol}, \gamma, (\text{foreign}, \text{symbol}, \gamma, \text{foreign}))$
9 **if** $(q \notin \mathbb{V} \wedge q' \notin \mathbb{V})$ **then**
10 $(Q_\gamma, \Sigma_\gamma, \Gamma_\gamma, E_\gamma) \xleftarrow{+} (\{q, q'\}, \sigma, \gamma, (q, \sigma, \gamma, q'))$
11 **if** $(q \in \mathbb{V} \wedge q' \notin \mathbb{V})$ **then**
12 $(Q_\gamma, \Sigma_\gamma, \Gamma_\gamma, E_\gamma) \xleftarrow{+} (q', \text{symbol}, \gamma, (\text{foreign}, \text{symbol}, \gamma, q'))$
13 **if** $(q \notin \mathbb{V} \wedge q' \in \mathbb{V})$ **then**
14 $(Q_\gamma, \Sigma_\gamma, \Gamma_\gamma, E_\gamma) \xleftarrow{+} (q, \text{symbol}, \gamma, (q, \text{symbol}, \gamma, \text{foreign}))$
15 **end**
16 **end**
17 **return** T_γ

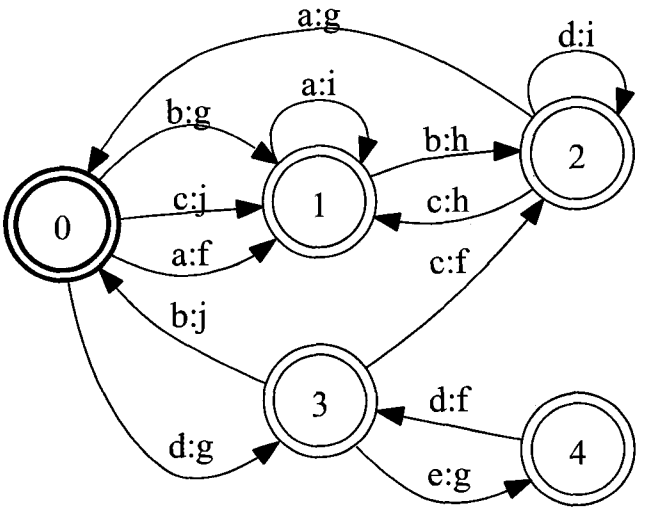

Figure 5.7: $T_\alpha \circ T_\beta \circ T_\gamma$ machine

Chapter 6

Method Evaluation

This chapter is concerned with evaluating the method proposed in chapter 5. A proof-of-concept is presented first, based on the service-based application developed in chapter 3. Then, time and space complexity analysis of the method is discussed.

6.1 Proof of Concept

This section outlines the representation of services as finite-state machines, and the application of algorithms presented in chapter 5.

6.1.1 Group Notification and Presence

One of the possible representations of a service that allows a group of users to send notification messages to each other or to the whole group, as well as being notified about new users joining the group or current users leaving the group is shown in figure 6.1. The service consists of only one state, in which it accepts requests to send messages and membership changes.

In this simplified model, the service is assumed to manage a list of current users and groups, and the users are assumed to fail gracefully, that is, by notifying the service of their departure. There are only four events to which the service will respond, a request to send a message to a particular user, a request to send a message to the whole group, a request to join the group and a request to leave the group. A listing of these events is given in table 6.1.

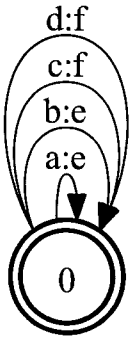

Figure 6.1: Model of group notification and presence service

6.1.2 Bulk Transfer

A user that wishes to transfer a large file to all members of the group, proceeds as follows. It first uploads a file to the service and receives a status of that operation. It then notifies the service that this file should be made available to the whole group. The user then notifies all members of the group that the file is available for download, using the `sendGroupMessage(msg)` functionality. All participants then download the file and notify the owner of the file of their succesful download using `sendPrivateMessage(msg)` functionality. The owner deletes the file from the service when all participants have downloaded the file. A listing of events specific to this service is given in table 6.2.

The service has three states, *ready*, *stored* and *downloading*. Time-outs (for example service duration specification) and file access control (basic security and group management) are assumed to be implemented by the service itself. Such a service could offer various performance characteristics, from storage space to download bandwidth made available to service consumers. A model of this service is shown in figure 6.2.

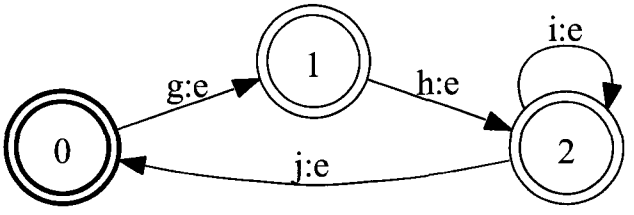

Figure 6.2: Model of bulk transfer service

Symbol	*Interpretation*
a	`sendPrivateMessage(msg)`: This functionality allows a message to be sent to a specific user. The `msg` parameter contains the unique identifier of the user and the message to be delivered.
b	`sendGroupMessage(msg)`: This functionality allows a message to be sent to all members of a specific group, including the sender. The `msg` parameter contains the unique identifier of the user and the message to be delivered.
c	`join(group)`: This functionality allows a user to become a member of a specific group.
d	`leave(group)`: This functionality allows a user to cease being a member of a specific group.
e	`status(msg)`: A user may receive a message detailing the status of some event. The `msg` parameter may be empty or contain information to be interpreted by the receiver.
f	`groupMembership(list)`: Whenever a user leaves or joins a group, this fact is made known to all current members of that group. The `list` parameter may contain the new membership list, or the difference with previously sent membership update.

Table 6.1: Group notification and presence service events

6.1.3 Data Consistency

All participants eventually must agree on the state of an object that can be modified by any one of them. A data consistency service then would help participants reach the consensus. A data consistency service may implement a strict exclusion or allow concurrent access to a given common object. In the model of figure 6.3 however, the data consistency service devolves the decision of allowing the modification to users themselves. This flexibility allows user to "implement" strict locking, majority voting, preemptive or some other mechanism. A listing of events specific to this service is given in table 6.3.

This service consists of two states, *unlocked* and *locked*. A specific object can only be modified when the service is in the locked state, a transition to which is decided by agreement of users. For example, if the users decide that the first request should be given the right to modify an object, then the timestamp of `modificationRequest(msg)` functionality would decide the user that is allowed to make modifications. If, on the other hand, majority voting is required, then users will send a request to modify, and

Symbol	*Interpretation*
g	`uploadFile(file)`: This functionality allows a file to be stored by the service.
h	`share(msg)`: This functionality allows a user to share a file where the `msg` parameter specifies a unique file identifier.
i	`download(msg)`: This functionality allows users to download a file where the `msg` parameter specifies a unique file identifier.
j	`delete(msg)`: This functionality allows a user to delete a previously uploaded file where the `msg` parameter specifies a unique file identifier.

Table 6.2: Events specific to the bulk transfer service

act based on the received replies.

Note that this service is modeled as a client of the presence service: the membership of the group information is used to make decisions (when the owner of the lock leaves for example) and to notify all interested users by changes to the common object. On an implementation level, this model can be achieved by either registering this service as a member of the group, or by having one member of the group forward it information about membership change.

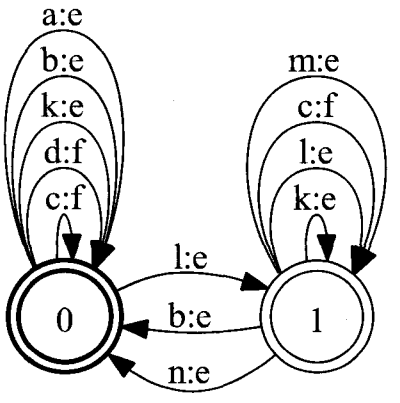

Figure 6.3: Model of data consistency service

Symbol	*Interpretation*
k	`modificationRequest(msg)`: This allows the user to request that an object specified by the `msg` parameter be modified.
l	`modify(msg)`: This allows the user to lock an object specified by the `msg` parameter.
m	`memberLeave()`: a group member left, where member $\neq$ lock owner
n	`ownerLeave()`: the lock owner left

Table 6.3: Events specific to the data consistency service

6.1.4 Exploiting Redundancy

Interaction with the services specified in figures 6.1, 6.2 and 6.3 allows the creation of a collaborative authoring application as discussed in chapter 3. If, however, we assume that an analysis of the incentives present in the data consistency service using the framework presented in chapter 4 yields a result that leads us to believe that this service is likely not to behave as specified, resulting in a negative impact on the reliability of the application, then we would like to replace it by a service based either on the notification and presence or the bulk transfer services. The assumption here, of course, is that those two candidate services are more reliable than the data consistency or target service.

It is easily found that there is no redundancy between the models of bulk transfer and data consistency services. On the other hand, application of algorithm 1 allows us to find the three submachines *alpha*, *beta* and *gamma*, shown in figures 6.4, 6.5 and 6.6 respectively. It can be easily verified that *alpha* $\circ$ *beta* $\circ$ *gamma* machine is equivalent to the data consistency service representation.

The *beta* submachine is a modified notification and presence service, modified in the sense that some new messages are defined and used between participants (e.g. $k0$: $e0$ message). Some of those new messages are allowed only in certain cases, which is simulated by another state (e.g. $k : k$ messages can only occur if $l0 : e1$ precedes it). Any implementation based on discussion in sections 5.5.1, 5.5.2 and 5.5.3 can be used in order to realize this submachine.

6.2 Complexity Evaluation

With the assumptions that $\Sigma_T = \Sigma_C$ and $|Q_C| \leq |Q_T|$, running the REDUNDANCY, CREATEALPHA, CREATEBETA and CREATEGAMMA algorithms, in the worst case, takes

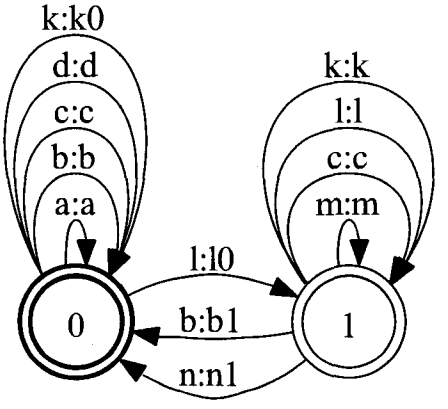

Figure 6.4: Machine alpha resulting from application of algorithm 5

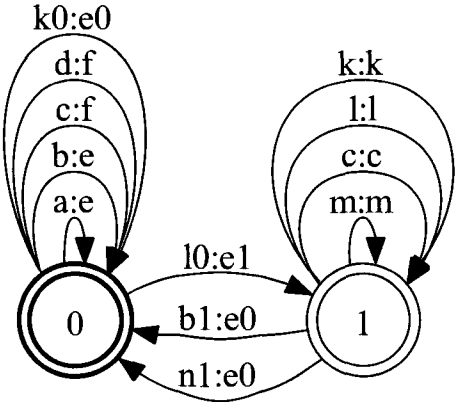

Figure 6.5: Machine beta resulting from application of algorithm 6

no more than $|Q_T| \times |Q_C| \times |Q_C| + |Q_T| \times |Q_C| \times |\Sigma_C|^2 + |Q_C| \times |\Sigma_C| + 3 \times |Q_T| \times |\Sigma_C|$ comparisons. Retaining dominant terms, the complexity of the proposed method depends on the number of states of both specifications and the size of the input alphabet and is of the order of $Q_T Q_C \Sigma_C (\frac{Q_C}{\Sigma_C} + \Sigma_C)$.

What is the consequence of using this method on the number of states and transitions? If the candidate specification is equivalent to the target specification ($C \equiv T$), then both T_α and T_γ machines will be simple identity machines, and no interpretation of T_β is necessary to realize it.

Considering the set of states and links of a machine, where the number of states and links of a machine M is denoted as $|M|$ and $< M >$ respectively, the number of states

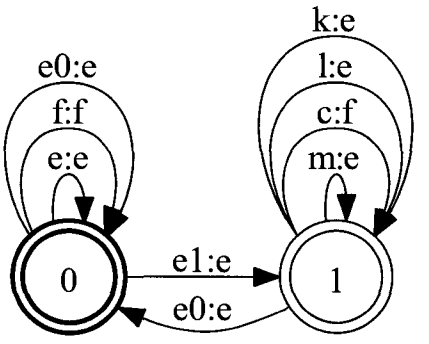

Figure 6.6: Machine gamma resulting from application of algorithm 7

and links for *alpha*, *beta* and *gamma* machines is as follows. *Alpha* will have one state and $< C >$ links (all functionality is performed by r state while s state is unconnected), *beta* is exactly the candidate machine C ($|C|$ states and $< C >$ links) and *gamma* has one state (foreign) and $< C >$ links. The total therefore is $|C| + 2$ states and $3 < C >$ links.

If there is no redundancy between the candidate and target machines, *alpha* will have one state and $< T >$ links (r state is unconnected), *beta* has one state (all functionality is performed by the foreign state) and $< T >$ links and *gamma* has $|T|$ states and $< T >$ links, resulting in a total of $|T| + 2$ states and $3 < T >$ links. This is of course comparable the previous case: either all work is done by the candidate machine or by the local implementation and control.[1]

For all cases in between, where some redundancy is found, *alpha* always has at most two states and at most $< T >$ links, *beta* has $|C| + 1$ states and at most $< C > + < T > - < C \cap T >$ links (where $< C \cap T >$ are the redundant transitions) and *gamma* has $|T| - |C \cap T| + 1$ states (where $|C \cap T|$ are the redundant states) and at most $< T >$ links. The total therefore is $|C| + |T| - |C \cap T| + 4$ states and $< C > + 3 < T > - < C \cap T >$ links. Realization of the *beta* machine will of course add to the space complexity, but is dependent on the specific implementation. The complexity of the proposed method therefore is polynomial with respect to time and linear with respect to space.

[1]In the case when $C \equiv T$, that is, T contains C and C contains T, then the number of states and links is exactly the same

Chapter 7

Conclusions

7.1 Summary of Results and Contributions

An example of an Internet collaborative application based on services is presented. Starting from the basic requirements of the application, it is shown how these can be fulfilled by utilizing certain services accessed through standard set of protocols and off-the-shelf techniques. If, however, a service is executed in a manner that is not consistent with its advertised performance, due to rational behavior of the service producer for example, then the application using this service might not meet user's requirements. If we can estimate which services are more likely to behave in this manner, then it would be possible to classify them into candidate and target services respectively. A qualitative reference framework is proposed which can be used for this classification.

In order to improve the robustness of the resulting application, it is desired to express target services in terms of candidate services, especially if the two services share some common functionality. The premise of services in the context of service-oriented architectures is that they may be used in contexts not foreseen by designers of those services. It is likely therefore that some of the functionality may be duplicated between services, or, in other words, that some part of a service will be redundant with respect to some part of another service. This thesis formalizes the concept of redundancy and proposes an algorithm to obtain this redundancy from any pair of services expressed as finite-state machines.

Using this concept of redundancy furthermore, an algorithm to express the target service in terms of a candidate service is proposed. The algorithm works by decomposing the target service into three submachines such that the entire functionality of the original

target machine can be recreated through application of the composition operation to the submachines. One of the submachines is a modified candidate machine while the remaining submachines represent the locally implemented functionality and control.

The performance of the proposed method is polynomial with respect to the service size (states and alphabet). When applied to increasing the reliability of an application however, the additional cost is the implementation of the non-redundant functionality and related control locally.

7.2 Applicability

Despite the best effort of its proponents, a wide variety of services available on the Internet is not going to come into existence at some discontinuity in the (near) future, but is more likely to appear progressively, with both the quantity and differentiation between services increasing only gradually as business cases are made, tried out and validated. The direct impact on the various automatic service composition algorithms is that it might not be able to replace a service with an equivalent one, or it might not be able to find enough of pertinent services to complete the composition. Furthermore, services that are available might initially be of an experimental nature or provided by technology enthusiasts without any warranty of the service quality, resulting in fragile compositions. On the upside, such composition of services has the potential to reduce the cost of creation, customization and maintenance of applications, and is therefore unlikely to be abandoned as long as the relentless quest to improve the bottom line continues. The reasons presenting a fairly negative case for applications built from services could thus be ascribed to the growing pains of a new technology, a period that most technologies go through and during which various proposals jostle for attention, evolve, and sometimes flourish and sometimes wither and fade.

In the short and medium term therefore, before there is an extant body of services, it might be useful to provide an algorithmic method that allows to construct applications from services, not so much by composition, but by reusing services to fill in the blank left by an unreliable or inexistent service.

7.3 Open Questions

The method presented in this thesis still leaves some open questions. First and foremost, the developed algorithms are applied to sequential finite-state machines (transducers)

which model services. Can all services be expressed in this way, or is there a class of services that can always be modeled by a transducer? Furthermore, can this model be arrived at in an "automatic" manner? Additionally, from the formalism point of view, can we characterize the class of transducers that lends itself to the described method (i.e. does this method apply to all sequential transducers)?

Secondly, algorithm 4 (`MAXG`) finds a local optimum. Is there a method that guarantees a global optimum? Current proposal expresses a target in terms of a candidate; can a target be expressed in terms of two or more candidates?

A potential improvement to the presented approach would allow to specify what constitutes redundancy for a user, allowing the preservation of certain properties across services. A limitation of this method is illustrated by the following scenario: two services, s_a and s_b use a third service s_c. Assuming service s_c is unreliable, a user may replace service s_a by s_b since from the point of view of the user, s_a is unreliable and services s_a and s_b share the common functionality s_c. In that case, the user would still be depending on the unreliable service.

Finally, the current service-based application was implemented with Jabber, JXTA and HTTP protocols. A WS-based implementation of the same application would allow to compare performance and ease of development, as well as potentially serve as a platform for future developments. Additionally, an algorithmic classification of services into target and candidate based on the INCA framework would help to make the proposed method fully automatic.

Bibliography

[1] *OASIS Reference Model for Service Oriented Architecture, Committee Draft 1.0, February 2006.*

[2] *Web Services Description Language (WSDL) 1.1.*

[3] *End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP)*, 2004.

[4] *OASIS Standards - Web Services Security v1.1*, 2006.

[5] Alfarez Abdul-Rahman and Stephen Hailes. Supporting trust in virtual communities. In *33rd Annual Hawaii International Conference on System Sciences (HICSS-33)*, 2000.

[6] Serge Abiteboul, Victor Vianu, Bradley S. Fordham, and Yelena Yesha. Relational transducers for electronic commerce. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*, pages 179–187. ACM Press, 1998.

[7] Eytan Adar and Bernardo Huberman. Free riding on gnutella. *FirstMonday*, 5(10), October 2000.

[8] Atul Adya, Robert Gruber, Barbara Liskov, and Umesh Maheshwari. Efficient optimistic concurrency control using loosely synchronized clocks. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 23–34. ACM Press, 1995.

[9] Deborah Agarwal, Keith Jackson, and Mary Thompson. Securing collaborative environments. Technical Report Paper LBNL-50427, Lawrence Berkeley National Laboratory, 2002.

[10] Deborah Agarwal, Markus Lorch, Mary Thompson, and Marcia Perry. A new security model for collaborative environments. Technical Report Paper LBNL-52894, Lawrence Berkeley National Laboratory, 2003.

[11] Rakesh Agrawal, Roberto Bayardo, Daniel Gruhl, and Spiros Papadimitriou. Vinci: a service-oriented architecture for rapid development of web applications. *Computer Networks*, 39(5):523–539, 2002.

[12] Atif Alamri, Mohamad Eid, and Abdulmotaleb El-Saddik. Classification of the state-of-the-art dynamic web services composition techniques. *International Journal of Web and Grid Services*, 2(2):148–166, 2006.

[13] Nazareno Andrade, Miranda Mowbray, Aliandro Lima, Gustavo Wagner, and Matei Ripeanu. Influences on cooperation in bittorrent communities. In *P2PECON '05: Proceeding of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 111–115, New York, NY, USA, 2005. ACM Press.

[14] Panayotis Antoniadis, Costas Courcoubetis, and Robin Mason. Comparing economic incentives in peer-to-peer networks. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 46(1):133–146, 2004.

[15] Robert Axelrod and William D. Hamilton. The evolution of cooperation. *Science*, 211:1390–1396, 1981.

[16] Magdalena Balazinska, Hari Balakrishnan, and Michael Stonebraker. Contract-based load management in federated distributed systems. In *NSDI*, pages 197–210. USENIX, 2004.

[17] G. Bamberg and K. Spremann. *Agency theory, Information and Incentives.* Springer, 1989.

[18] Marie-Pierre Béal and Olivier Carton. Determinization of transducers over finite and infinite words. *Journal of Theoretical Computer Science*, 289(1):225–251, October 2002.

[19] Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Journal of Theoretical Computer Science*, 292(1):45–63, January 2003.

[20] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. Automatic composition of e-services that export their behavior. In Maria E. Orlowska, Sanjiva Weerawarana, Mike P. Papazoglou, and Jian Yang, editors, *ICSOC*, volume 2910 of *Lecture Notes in Computer Science*, pages 43–58. Springer, 2003.

[21] Daniel S. Bernstein, Zhengzhu Feng, Brian Neil Levine, and Shlomo Zilberstein. Adaptive peer selection. In Kaashoek and Stoica [74], pages 237–246.

[22] Philip A. Bernstein and Nathan Goodman. A sophisticate's introduction to distributed concurrency control (invited paper). In *Eigth International Conference on Very Large Data Bases, September 8-10, 1982, Mexico City, Mexico, Proceedings*, pages 62–76. Morgan Kaufmann, 1982.

[23] Elisa Bertino, Elena Ferrari, and Vijayalakshmi Atluri. A flexible model supporting the specification and enforcement of role-based authorization in workflow management systems. In *ACM Workshop on Role-Based Access Control*, pages 1–12, 1997.

[24] Ranjita Bhagwan, Stefan Savage, and Geoffrey M. Voelker. Understanding availability. In Kaashoek and Stoica [74], pages 256–267.

[25] Ashwin R. Bharambe, Cormac Herley, and Venkata N. Padmanabhan. Analyzing and improving a bittorrent network's performance mechanisms. In *IEEE INFOCOM*, 2006.

[26] Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. Verifying policy-based security for web services. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick Drew McDaniel, editors, *ACM Conference on Computer and Communications Security*, pages 268–277. ACM, 2004.

[27] P. Bhoj, S. Singhal, and S. Chutani. SLA management in federated environments. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 35(1):5–24, 2001.

[28] Egon Börger. Modeling workflow patterns from first principles. In Christine Parent, Klaus-Dieter Schewe, Veda C. Storey, and Bernhard Thalheim, editors, *ER*, volume 4801 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2007.

[29] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[30] F. Bukhari and Sylvia L. Osborn. Two fully distributed concurrency control algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 5(5):872–881, 1993.

[31] Tevfik Bultan, Xiang Fu, Richard Hull, and Jianwen Su. Conversation specification: a new approach to design and analysis of e-service composition. In *WWW*, pages 403–410, 2003.

[32] Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. A game theoretic framework for incentives in p2p systems. In Shahmehri et al. [131], pages 48–56.

[33] Mark H. Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew V. McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne, and Katia P. Sycara. Daml-s: Web service description for the semantic web. In Ian Horrocks and James A. Hendler, editors, *International Semantic Web Conference*, volume 2342 of *Lecture Notes in Computer Science*, pages 348–363. Springer, 2002.

[34] Ran Canetti, Juan A. Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Proceedings IEEE INFOCOM '99*, volume 2, pages 708–716, 1999.

[35] Germano Caronni and Marcel Waldvogel. Establishing trust in distributed storage providers. In Shahmehri et al. [131], pages 128–133.

[36] Jo-Mei Chang and Nicholas F. Maxemchuk. Reliable broadcast protocols. *ACM Transactions on Computer Systems*, 2:251–273, 1984.

[37] Christian Choffrut. Une caracterisation des fonctions sequentielles et des fonctions sous-sequentielles en tant que relations rationnelles. *Journal of Theoretical Computer Science*, 5(3):325–337, 1977.

[38] Christian Choffrut. Minimizing subsequential transducers: a survey. *Journal of Theoretical Computer Science*, 292(1):131–143, 2003.

[39] Bram Cohen. Incentives build robustness in BitTorrent. In *1st Workshop on Economics of Peer-to-Peer Systems (P2PECON), Berkeley, California*, 2003.

[40] Tyson Condie, Sepandar D. Kamvar, and Hector Garcia-Molina. Adaptive peer-to-peer topologies. In Germano Caronni, Nathalie Weiler, and Nahid Shahmehri, editors, *Peer-to-Peer Computing*, pages 53–62. IEEE Computer Society, 2004.

[41] Domenico Cotroneo, Almerindo Graziano, and Stefano Russo. Security requirements in service oriented architectures for ubiquitous computing. In Paddy Nixon and Fabio Kon, editors, *Middleware for Pervasive and Ad-hoc Computing*, pages 172–177. ACM, 2004.

[42] Zhe Dang, Oscar H. Ibarra, and Jianwen Su. Composability of infinite-state activity automata. In Rudolf Fleischer and Gerhard Trippen, editors, *ISAAC*, volume 3341 of *Lecture Notes in Computer Science*, pages 377–388. Springer, 2004.

[43] Prashant Dewan. Peer-to-peer reputations. In *IPDPS*. IEEE Computer Society, 2004.

[44] Hans-Peter Dommel and J. J. Garcia-Luna-Aceves. Efficacy of floor control protocols in distributed multimedia collaboration. *Cluster Computing*, 2(1):17–33, 1999.

[45] John R. Douceur. The sybil attack. In Druschel et al. [46], pages 251–260.

[46] Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors. *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, volume 2429 of *Lecture Notes in Computer Science*. Springer, 2002.

[47] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *International Journal of Web and Grid Services*, 1(1):1–30, 2005.

[48] Quazi Ehsanul, Kabir Mamun, and Hidenori Nakazato. A new token based protocol for group mutual exclusion in distributed systems. In *ISPDC*, pages 34–41. IEEE Computer Society, 2006.

[49] Jeremy Epstein, Scott Matsumoto, and Gary McGraw. Software security and soa: danger, will robinson! *IEEE Security and Privacy Magazine*, 4(1):80–83, Jan.-Feb. 2006.

[50] Roozbeh Farahbod, Uwe Glässer, and Mona Vajihollahi. Specification and validation of the business process execution language for web services. In Wolf Zimmer-

mann and Bernhard Thalheim, editors, *Abstract State Machines*, volume 3052 of *Lecture Notes in Computer Science*, pages 78–94. Springer, 2004.

[51] Joan Feigenbaum and Scott Shenker. Distributed algorithmic mechanism design: recent results and future directions. In *DIAL-M*, pages 1–13. ACM, 2002.

[52] Michal Feldman, Kevin Lai, Ion Stoica, and John Chuang. Robust incentive techniques for peer-to-peer networks. In Jack S. Breese, Joan Feigenbaum, and Margo I. Seltzer, editors, *ACM Conference on Electronic Commerce*, pages 102–111. ACM, 2004.

[53] Andrea Ferrara. Web services: a process algebra approach. In Marco Aiello, Mikio Aoyama, Francisco Curbera, and Mike P. Papazoglou, editors, *ICSOC*, pages 242–251. ACM, 2004.

[54] Christof Fetzer and Flaviu Cristian. An optimal internal clock synchronization algorithm. In *Proceedings of the Tenth Annual Conference on Computer Assurance, 'Systems Integrity, Software Safety and Process Security' (Compass '95)*, pages 187–196, Gaithersburg, MD, USA, 1995.

[55] E. J. Friedman and P. Resnick. Social cost of cheap pseudonyms. *Journal of Economics and Management Strategy*, 10:173–199, 2001.

[56] Tamás Gaál. Is this finite-state transducer sequentiable? In Bruce W. Watson and Derick Wood, editors, *CIAA*, volume 2494 of *Lecture Notes in Computer Science*, pages 125–134. Springer, 2001.

[57] Rosario Gennaro and Pankaj Rohatgi. How to sign digital streams. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 180–197. Springer, 1997.

[58] Dipak Ghosal, Benjamin K. Poon, and Keith Kong. P2P contracts: a framework for resource and service exchange. *Future Gener. Comput. Syst.*, 21(3):333–347, 2005.

[59] Arthur Gill. *Introduction to the Theory of Finite-State Machines.* McGraw-Hill Book Company, 1962.

[60] Philippe Golle, Kevin Leyton-Brown, and Ilya Mironov. Incentives for sharing in peer-to-peer networks. In *ACM Conference on Electronic Commerce*, pages 264–267. ACM, 2001.

[61] Wolfgang Grieskamp, Yuri Gurevich, Wolfram Schulte, and Margus Veanes. Generating finite state machines from abstract state machines. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 112–122, 2002.

[62] Yuri Gurevich. Sequential abstract-state machines capture sequential algorithms. *ACM Trans. Comput. Log.*, 1(1):77–111, 2000.

[63] Ahsan Habib and John Chuang. Incentive mechanism for peer-to-peer media streaming. In *IWQoS*, pages 171–180. IEEE, 2004.

[64] Rachid Hamadi and Boualem Benatallah. A petri net-based model for web service composition. In Klaus-Dieter Schewe and Xiaofang Zhou, editors, *ADC*, volume 17 of *CRPIT*, pages 191–200. Australian Computer Society, 2003.

[65] Seyyed Vahid Hashemian and Farhad Mavaddat. A graph-based framework for composition of stateless web services. In *ECOWS*, pages 75–86. IEEE Computer Society, 2006.

[66] David Hausheer, Nicolas Liebau, Andreas Mauthe, Ralf Steinmetz, and Burkhard Stiller. Token-based accounting and distributed pricing to introduce market mechanisms in a peer-to-peer file sharing scenario. In Shahmehri et al. [131], pages 200–201.

[67] Frederick C. Hennie. *Finite-State Models for Logical Machines.* John Wiley & Sons, Inc., 1968.

[68] Gerard J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, 1991.

[69] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 2nd edition, 2001.

[70] Thomas Hummel, Øyvind Strømme, and Ryan M. La Salle. Earning a living among peers - the quest for viable p2p revenue models. In *36th Hawaii International Conference on System Sciences (HICSS-36 2003)*, page 219, 2003.

[71] Henning Qin Jehøj, Niels Olof Bouvin, and Kaj Grønbæk. Awaredav: a generic webdav notification framework and implementation. In Allan Ellis and Tatsuya Hagino, editors, *WWW*, pages 180–189. ACM, 2005.

[72] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, 2007.

[73] Seung Jun and Mustaque Ahamad. Incentives in bittorrent induce free riding. In *P2PECON '05: Proceeding of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 116–121, New York, NY, USA, 2005. ACM Press.

[74] M. Frans Kaashoek and Ion Stoica, editors. *Peer-to-Peer Systems, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22,2003, Revised Papers*, volume 2735 of *Lecture Notes in Computer Science*. Springer, 2003.

[75] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the Twelfth International World Wide Web Conference (WWW2003)*, pages 640–651, Budapest, Hungary, May 2003.

[76] Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.

[77] Lauri Karttunen and Kenneth R. Beesley. *Inquiries into Words, Constraints and Contexts. Festschrift for Kimmo Koskenniemi on his 60th Birthday (2005)*, chapter Twenty-five years of finite-state morphology, pages 71–83. CSLI Studies in Computational Linguistics ONLINE, Copestake, Ann (Series Editor). CSLI Publications, Stanford, California, 2005.

[78] Tomomi Kawashima and Jianhua Ma. Tomscop - a synchronous p2p collaboration platform over jxta. In *ICDCS Workshops*, pages 85–90. IEEE Computer Society, 2004.

[79] Alexander Keller and Heiko Ludwig. The WSLA framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11(1):57–81, 2003.

[80] André Kempe. Factorization of ambiguous finite-state transducers. In Yu and Paun [157], pages 170–181.

[81] Efim Kinber and Carl Smith. *Theory of Computing, A Gentle Introduction.* Prentice-Hall, 2001.

[82] Ramayya Krishnan, Michael D. Smith, Zhulei Tang, and Rahul Telang. The impact of free-riding on peer-to-peer networks. In *37th Hawaii International Conference on System Sciences (HICSS-37 2004)*, 2004.

[83] Ajay D. Kshemkalyani. The power of logical clock abstractions. *Distributed Computing*, 17(2):131–150, 2004.

[84] Andreas Kuehlmann and Cornelis A. J. van Eijk. Combinational and sequential equivalence checking. In Soha Hassoun and Tsutomu Sasao, editors, *Logic Synthesis and Verification*, pages 343–372. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[85] M. Adlerand R. Kumar, K. Ross, D. Rubenstein, D. Turner, and D. Yao. Optimal peer selection in a free-market peer-resource economy. In *Second Workshop on the Economics of Peer-to-Peer Systems, Harvard University*, 2004.

[86] K. Lai, M. Feldman, I. Stoica, and J. Chuang. Incentives for cooperation in peer-to-peer networks. In *Workshop on Economics of Peer-to-Peer Systems, Berkeley, USA*, 2003.

[87] Nathaniel Leibowitz, Matei Ripeanu, and Adam Wierzbicki. Deconstructing the kazaa network. In *WIAPP '03: Proceedings of the The Third IEEE Workshop on Internet Applications*, page 112, Washington, DC, USA, 2003. IEEE Computer Society.

[88] Brian Neil Levine and J. J. Garcia-Luna-Aceves. A comparison of reliable multicast protocols. *Multimedia Systems*, 6(5):334–348, 1998.

[89] Brian Neil Levine, Clay Shields, and N. Boris Margolin. A survey of solutions to the sybil attack. Technical Report 2006-052, University of Massachusetts Amherst, October 2006.

[90] Heiko Ludwig and Charles J. Petrie. 05462 session summary - "cross cutting concerns". In Francisco Curbera, Bernd J. Krämer, and Mike P. Papazoglou, editors,

Service Oriented Computing, volume 05462 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[91] Jennifer Lundelius and Nancy A. Lynch. A new fault-tolerant algorithm for clock synchronization. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 75–88, 1984.

[92] Richard T. B. Ma, Sam C. M. Lee, John C. S. Lui, and David K. Y. Yau. A game theoretic approach to provide incentive and service differentiation in p2p networks. In Edward G. Coffman Jr., Zhen Liu, and Arif Merchant, editors, *SIGMETRICS*, pages 189–198. ACM, 2004.

[93] Wenbo Mao, Fei Yan, and Chunrun Chen. Daonity: grid security with behaviour conformity from trusted computing. In *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*, pages 43–46, New York, NY, USA, 2006. ACM Press.

[94] Sergio Marti and Hector Garcia-Molina. Identity crisis: Anonymity vs. reputation in p2p systems. In Shahmehri et al. [131], pages 134–141.

[95] Martin Mauve, Volker Hilt, Christoph Kuhmnch, and Wolfgang Effelsberg. RTP/I - Toward a Common Application Level Protocol for Distributed Interactive Media. *IEEE Transactions on Multimedia*, 3(1):152–161, March 2001.

[96] Nikola Milanovic and Miroslaw Malek. Current solutions for web service composition. *IEEE Internet Computing*, 8(6):51–59, 2004.

[97] David L. Mills. A brief history of ntp time: memoirs of an internet timekeeper. *Computer Communication Review*, 33(2):9–21, 2003.

[98] Mehryar Mohri. Compact representations by finite-state transducers. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 204–209. Association for Computational Linguistics, 1994.

[99] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

[100] Mehryar Mohri. Minimization algorithms for sequential transducers. *Journal of Theoretical Computer Science*, 234(1-2):177–201, 2000.

[101] Gianluca Moro, Claudio Sartori, and Munindar P. Singh, editors. *Agents and Peer-to-Peer Computing, Second International Workshop, AP2PC 2003, Melbourne, Australia, July 14, 2003, Revised and Invited Papers*, volume 2872 of *Lecture Notes in Computer Science*. Springer, 2004.

[102] Jonathan P. Munson and Prasun Dewan. A concurrency control framework for collaborative systems. In *Proceedings of the ACM 1996 Conference on Computer Supported Cooperative Work (CSCW '96)*, pages 278–287, Boston, MA, USA., November 16-20 1996. ACM.

[103] Chaki Ng, David C. Parkes, and Margo Seltzer. Strategyproof computing: Systems infrastructures for self-interested parties. In *Workshop on Economics of Peer-to-Peer Systems, Berkeley, USA*, 2003.

[104] Tsuen-Wan Ngan, Dan S. Wallach, and Peter Druschel. Enforcing fair sharing of peer-to-peer resources. In Kaashoek and Stoica [74], pages 149–159.

[105] Seth James Nielson, Scott Crosby, and Dan S. Wallach. A taxonomy of rational attacks. In Miguel Castro and Robbert van Renesse, editors, *IPTPS*, volume 3640 of *Lecture Notes in Computer Science*, pages 36–46. Springer, 2005.

[106] OASIS. *Web Services Business Process Execution Language v2.0*, April 2007.

[107] Katia Obraczka. Multicast transport protocols: a survey and taxonomy. *IEEE Communications Magazine*, 1(36):94–102, 1998.

[108] Philipp Obreiter and Jens Nimis. A taxonomy of incentive patterns - the design space of incentives for cooperation. In Moro et al. [101], pages 89–100.

[109] Venkata N. Padmanabhan, Helen J. Wang, Philip A. Chou, and Kunwadee Sripanidkulchai. Distributing streaming media content using cooperative networking. In *NOSSDAV*, pages 177–186. ACM, 2002.

[110] Mike P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *WISE*, pages 3–12. IEEE Computer Society, 2003.

[111] Charles J. Petrie and Christoph Bussler. Service agents and virtual enterprises: A survey. *IEEE Internet Computing*, 7(4):68–78, 2003.

[112] George V. Popescu and Zhen Liu. Stateless application-level multicast for dynamic group communication. In *DS-RT*, pages 20–28. IEEE Computer Society, 2004.

[113] Dongyu Qiu and Rayadurgam Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In Raj Yavatkar, Ellen W. Zegura, and Jennifer Rexford, editors, *SIGCOMM*, pages 367–378. ACM, 2004.

[114] Kavitha Ranganathan, Matei Ripeanu, Ankur Sarin, and Ian T. Foster. To share or not to share, an analysis of incentives to contribute in collaborative file sharing environments. In *Workshop on Economics of Peer-to-Peer Systems, Berkeley, USA*, 2003.

[115] Jinghai Rao, Peep Küngas, and Mihhail Matskin. Composition of semantic web services using linear logic theorem proving. *Information Systems*, 31(4):340–360, 2006.

[116] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In Jorge Cardoso and Amit P. Sheth, editors, *SWSWPC*, volume 3387 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2004.

[117] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. Reputation systems. *Communications of the ACM (CACM)*, 43(12):45–48, 2000.

[118] Glenn Ricart and Ashok K. Agrawala. An optimal algorithm for mutual exclusion in computer networks. *Communications of the ACM (CACM)*, 24(1):9–17, 1981.

[119] Emmanuel Roche. Compact factorization of finite-state transducers and finite-state automata. *Nord. J. Comput.*, 4(2):187–216, 1997.

[120] Emmanuel Roche and Yves Schabes. Deterministic part-of-speech tagging with finite-state transducers. *Computational Linguistics*, 21(2):227–253, 1995.

[121] Emmanuel Roche and Yves Shabes, editors. *Finite-State Language Processing.* MIT Press, Cambridge, MA, USA, 1997.

[122] Don Ross. Game theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy.* The Metaphysics Research Lab, Center for the Study of Language and Information, Stanford University, Stanford, CA 94305-4115, Winter 2004.

[123] Lawrence A. Rowe and Ramesh Jain. Acm sigmm retreat report on future directions in multimedia research. *Transactions on Multimedia Computing, Communications and Applications (TOMCCAP)*, 1(1):3–13, 2005.

[124] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia Systems*, 9(2):170–184, 2003.

[125] André Schiper, Kenneth Birman, and Pat Stephenson. Lightweight causal and atomic group multicast. *ACM Trans. Comput. Syst.*, 9(3):272–314, 1991.

[126] Eve M. Schooler. Conferencing and collaborative computing. *Multimedia Systems*, 4(5):210–225, 1996.

[127] Thorsten Schütt, Florian Schintke, and Alexander Reinefeld. Efficient synchronization of replicated data in distributed systems. In Peter M.A. Sloot, David Abramson, Alexander V. Bogdanov, Jack J. Dongarra, Albert Y. Zomaya, and Yuriy E. Gorbachev, editors, *Computational Science - ICCS 2003, International Conference Melbourne, Australia and St. Petersburg, Russia, Proceedings, Part I*, volume 2657 of *Lecture Notes in Computer Science*, pages 274–283. Springer, June 2003.

[128] Marcel Paul Schützenberger. Sur les relations rationnelles. In H. Barkhage, editor, *Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science 646589*, pages 209–213. Springer, 1975.

[129] Mark Senior and Ralph Deters. Market structures in peer computation sharing. In Ross Lee Graham and Nahid Shahmehri, editors, *Peer-to-Peer Computing*, pages 128–135. IEEE Computer Society, 2002.

[130] Andrei Serjantov and Stephen Lewis. Puzzles in p2p systems. In *8th CaberNet Radicals Workshop, Corsica*, 2003.

[131] Nahid Shahmehri, Ross Lee Graham, and Germano Caronni, editors. *3rd International Conference on Peer-to-Peer Computing (P2P 2003), 1-3 September 2003, Linköping, Sweden.* IEEE Computer Society, 2003.

[132] Haifeng Shen and Chengzheng Sun. Flexible notification for collaborative systems. In *Proceeding on the ACM Conference on Computer Supported Cooperative Work (CSCW)*, pages 77–86, New Orleans, Louisiana, USA., November 16-20 2002.

[133] Jeffrey Shneidman, David Parkes, and Laurent Massoulié. Faithfulness in internet algorithms. In *Proceedings of SIGCOMM Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems (PINS'04), Portland, Oregon*, pages 220–227, New York, NY, USA, 2004. ACM Press.

[134] Jeffrey Shneidman and David C. Parkes. Rationality and self-interest in peer to peer networks. In Kaashoek and Stoica [74], pages 139–148.

[135] Aameek Singh and Ling Liu. Trustme: Anonymous management of trust relationships in decentralized p2p systems. In Shahmehri et al. [131], pages 142–149.

[136] Ben Strulo, Alan Smith, and Jeff Farr. An architecture for peer-to-peer economies. In Shahmehri et al. [131], pages 208–209.

[137] Chengzheng Sun and David Chen. Consistency maintenance in real-time collaborative graphics editing systems. *ACM Transactions on Computer-Human Interaction*, 9(1):1–41, March 2002.

[138] Mee Young Sung. A collaborative multimedia authoring system based on the conceptual temporal relations. In Kiyoharu Aizawa, Yuichi Nakamura, and Shin'ichi Satoh, editors, *PCM (2)*, volume 3332 of *Lecture Notes in Computer Science*, pages 649–656. Springer, 2004.

[139] G. Suryanarayana and R.N. Taylor. A survey of trust management and resource discovery technologies in peer-to-peer applications. Technical Report Technical Report UCI-ISR-04-6, UCI Institute for Software Research, 2004.

[140] Maurice ter Beek, Antonio Bucchiarone, and Stefania Gnesi. Web service composition approaches: From industrial standards to formal methods. In *International Conference on Internet and Web Applications and Services (ICIW'07)*, page 15, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[141] L. Terveen and W. Hill. Beyond recommender systems: Helping people help each other. In John M. Carroll, editor, *Human-computer interaction in the new millennium*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2001.

[142] Alexander Thomasian. Two-phase locking performance and its thrashing behavior. *ACM Transactions on Database Systems*, 18(4):579–625, 1993.

[143] Trusted Computing Group. *Infrastructure Specifications*, February 2007.

[144] Amin Vahdat, Jeffrey S. Chase, Rebecca Braynard, Dejan Kostic, Patrick Reynolds, and Adolfo Rodriguez. Self-organizing subsets: From each according to his abilities, to each according to his needs. In Druschel et al. [46], pages 76–84.

[145] W.M.P. van der Aalst. Pi calculus versus petri nets: Let us eat humble pie rather than further inflate the pi hype. Unpublished discussion paper, 2003.

[146] Sudharshan S. Vazhkudai, Xiaosong Ma, Vincent W. Freeh, Jonathan W. Strickland, Nandan Tammineedi, and Stephen L. Scott. Freeloader: Scavenging desktop storage resources for scientific data. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 56, Washington, DC, USA, 2005. IEEE Computer Society.

[147] Marc Waldman and David Mazières. Tangler: a censorship-resistant publishing system based on document entanglements. In Tomas Sander, editor, *ACM Conference on Computer and Communications Security*, volume 2320 of *Lecture Notes in Computer Science*, pages 126–135. Springer, 2001.

[148] Dan S. Wallach. A survey of peer-to-peer security issues. In Mitsuhiro Okada, Benjamin C. Pierce, Andre Scedrov, Hideyuki Tokuda, and Akinori Yonezawa, editors, *ISSS*, volume 2609 of *Lecture Notes in Computer Science*, pages 42–57. Springer, 2002.

[149] Weihong Wang and Baochun Li. To play or to control: A game-based control-theoretic approach to peer-to-peer incentive engineering. In Kevin Jeffay, Ion Stoica, and Klaus Wehrle, editors, *IWQoS*, volume 2707 of *Lecture Notes in Computer Science*, pages 174–194. Springer, 2003.

[150] Yao Wang and Julita Vassileva. Bayesian network trust model in peer-to-peer networks. In Moro et al. [101], pages 23–34.

[151] Todd Wareham. The parameterized complexity of intersection and composition operations on sets of finite-state automata. In Yu and Paun [157], pages 302–310.

[152] Brian Whetten, Todd Montgomery, and Simon M. Kaplan. A high performance totally ordered multicast protocol. In Kenneth P. Birman, Friedemann Mattern, and André Schiper, editors, *Dagstuhl Seminar on Distributed Systems*, volume 938 of *Lecture Notes in Computer Science*, pages 33–57. Springer, 1994.

[153] J. Widmer, R. Denda, and M. Mauve. A survey on TCP-friendly congestion control. *IEEE Network*, 15(3):28–37, 2001.

[154] Chung Kei Wong, Mohamed G. Gouda, and Simon S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, 8(1):16–30, 2000.

[155] Chung Kei Wong and Simon S. Lam. Digital signatures for flows and multicasts. *IEEE/ACM Transactions on Networking (TON)*, 7(4):502–513, 1999.

[156] Pinar Yolum and Munindar P. Singh. Engineering self-organizing referral networks for trustworthy service selection. *IEEE Transactions on System, Man, and Cybernetics, Part A: Systems and Humans (TSMC)*, 35(3):396–407, 2005.

[157] Sheng Yu and Andrei Paun, editors. *Implementation and Application of Automata, 5th International Conference, CIAA 2000, London, Ontario, Canada, July 24-25, 2000, Revised Papers*, volume 2088 of *Lecture Notes in Computer Science*. Springer, 2001.

[158] Rongmei Zhang and Y. Charlie Hu. Borg: a hybrid protocol for scalable application-level multicast in peer-to-peer networks. In Christos Papadopoulos and Kevin C. Almeroth, editors, *NOSSDAV*, pages 172–179. ACM, 2003.

Appendix A

Finite-State Machines

A.1 Finite State Machines

This chapter presents a brief summary of the finite-state machines, along with some definitions that will be used in subsequent chapters.

A.1.1 Deterministic Machines

A finite state machine (FSM) can be defined [59] as:

Definition A.1.1. *(FSM) A finite state machine M is a synchronous system represented by a quintuple $M = (\Sigma, \Gamma, Q, \delta, \omega)$ with a finite input alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_i\}$, a finite output alphabet $\Gamma = \{\gamma_1, \ldots, \gamma_j\}$, a finite state set $Q = \{q_1, \ldots, q_n\}$ and a pair of characterizing functions δ and ω given by $q_{v+1} = \delta(q_v, \sigma_v)$, $\gamma_v = \omega(q_v, \sigma_v)$ where σ_v, γ_v, q_v are the input symbol, output symbol and state of M at $v = 1, 2, \ldots$*

Thus the input to the machine is a sequence of symbols $\sigma_1 \ldots \sigma_k$, and the output of a machine is a sequence of symbols $\gamma_1 \ldots \gamma_k$. In general then, we can describe the behavior of a machine that is in some state in response to some input sequence using the following definitions:

Definition A.1.2. *(Machine configuration) The configuration (q, s) of a machine $M = (\Sigma, \Gamma, Q, \delta, \omega)$ is any element of $Q \times \Sigma^*$.*

Where Σ^* is the set of all strings over the alphabet Σ. This is defined in terms of the powers of the alphabet, which defines sets of strings formed from the alphabet with

specific lengths. The empty string (absence of symbols) is denoted as ϵ. Thus, $\Sigma^0 = \{\epsilon\}$, $\Sigma^1 = \Sigma$, $\Sigma^2 = \Sigma \times \Sigma$ and so on, giving $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots$

Definition A.1.3. *(Machine yield) A machine in configuration (q, s) yields the configuration (q', s') in one step if there exist a symbol $\sigma \in \Sigma$ such that $s = \sigma s'$ and a transition function $\delta(q, \sigma) = q'$.*

Where we have settled on a "direction" in the sense that the machine "consumes" one symbol at a time from the string instead of "produces" longer strings which would be the case if $s' = \sigma s$.

Noting that if $\delta(q, \epsilon) = q$, that is, for an empty string input the state does not change, it can be shown by induction [69] that a configuration (q_1, s_1) yields a configuration (q_n, s_n) if there exists a sequence of configurations (q_1, s_1), (q_2, s_2),..., (q_n, s_n) where (q_i, s_i) yields (q_{i+1}, s_{i+1}) in one step. We can thus define an extended transition function [69] which takes a state and a string and returns a state p after processing the string s starting from state q:

$$\hat{\delta}(q, s) = p \tag{A.1}$$

We can similarly define the extended output function which takes an input string and returns an output string by setting $\omega(q, \epsilon) = \epsilon$ and where $\forall \sigma \in \Sigma$, $\forall s \in \Sigma^*$, $s = \sigma s'$

$$\hat{\omega}(q, s) = \hat{\omega}(q, \sigma s') = \omega(q, \sigma)\hat{\omega}(\delta(q, \sigma), s') = s'' \tag{A.2}$$

To predict the output sequence produced by a machine, knowledge of the input sequence and the functions δ, ω is not sufficient and the initial state (the state of the machine when the input sequence is presented) is also required. Additionally, an acceptance condition can be specified (the machine must end in some predefined state for example for the result to be valid) and represented as a subset of the valid states. For these reasons, an alternate and more specialized definition is given as:

Definition A.1.4. *(DFSM) A deterministic finite state machine M is a tuple $M = (\Sigma, \Gamma, Q, q_0, \delta, \omega, F)$ where Σ is a finite input alphabet, Γ is a finite output alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state of the machine, δ is a transition function, ω is the output function and F represents the acceptance condition where $F \subseteq Q$.*

A further distinction can be made for the output function ω which can be expressed as either:

$$\omega : Q \times \Sigma \rightarrow \Gamma \tag{A.3}$$

$$\omega : Q \rightarrow \Gamma \tag{A.4}$$

describing a transition-assigned (or Mealy) machine or a state-assigned (or Moore) machine respectively. The exact choice between the Mealy or Moore machines is usually driven by the application at hand and systems that can be represented by one machine can also be represented by the other[1]. One of the differences is the amount of states necessary to represent the system, however, as expressed in [59]:

> *The selection of a state set in any given problem is neither unique nor necessarily simple. Since no general rules are available for this selection, it often calls for a trial-and-error approach. The speed at which the state set is selected and the size of the selected set are, by and large, a matter of "insight" and personal acquaintance with the system at hand.*

Both Mealy and Moore machines are just special cases of more general machines where each input symbol may give an output sequence (e.g., $\omega : Q \times \Sigma \rightarrow \Gamma^*$), or an input sequence gives an output sequence (e.g., $\omega : Q \times \Sigma^* \rightarrow \Gamma^*$).

Note that the machine as defined in Definition A.1.4 can only be in one state at a time, and has the following property: $\forall p, q, r \in Q$, $\forall \sigma \in \Sigma$, $\forall \gamma, \gamma' \in \Gamma$, if $\delta(p, \sigma) = r$, $\omega(p, \sigma) = \gamma$ and $\delta(q, \sigma) = r$, $\omega(q, \sigma) = \gamma'$ then $p = q$ and $\gamma = \gamma'$ (or if we assume that equation $\delta(q, \sigma)$ returns a set of states, then this property will be simply $|\delta(q, \sigma)| \leq 1$, $\forall q, \sigma$) A nondeterministic machine on the other hand can be in multiple states at once, and as such is a conceptual tool and does not represent a physical implementation of some system or behavior.

A.1.2 Nondeterministic Machines

In contrast to the deterministic machines, nondeterministic machines may for instance change state even if there is no input, may have multiple starting states and there may be none or more executable transition rules. To accommodate those differences a transition mapping is defined as $\delta_n : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$, where the return value is represented as a powerset[2] of Q. The output mapping can similarly be defined as $\omega_n : Q \times \Sigma^* \times Q \rightarrow \Gamma^*$. In the nondeterministic cases, an alternative notation is more convenient, and instead of the mappings δ_n and ω_n, the transition relation Δ is used:

[1]Note that in the case of the Moore machine, an output could be generated even in the absence of an input (empty string ϵ). The Mealy machine cannot do that unless $\epsilon \in \Sigma$.

[2]A powerset of S is the set of all subsets of S, including the empty set $\emptyset$

Definition A.1.5. *(NFSM) A nondeterministic finite state machine M is a tuple $M = (\Sigma, \Gamma, Q, Q_0, \Delta, F)$ where Σ is a finite input alphabet, Γ is a finite output alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ are the initial states of the machine, $\Delta \subset Q \times \Sigma^* \times \Gamma^* \times Q$ is the transition relation and F represent the acceptance condition where $F \subseteq Q$.*

Each transition relation $(p, \sigma, \gamma, q) \in \Delta$, or edge from state p to state q can be represented as $p \xrightarrow{\sigma|\gamma} q$, where σ and γ play the role of input and output labels of the edge respectively. Each input to the machine will describe a *path* which is a finite sequence [18] of relations represented as: $q_0 \xrightarrow{\sigma_1|\gamma_1} q_1 \xrightarrow{\sigma_2|\gamma_2} q_2 \cdots \xrightarrow{\sigma_n|\gamma_n} q_n$. The input and output labels of the path are $\sigma_{in} = \sigma_1\sigma_2 \ldots \sigma_n$ and $\gamma_{out} = \gamma_1\gamma_2 \ldots \gamma_n$ respectively. The machine thus computes the relation $(\sigma_{in}, \gamma_{out})$ on $\Sigma^* \times \Gamma^*$. A path is called *successful* if it starts from an initial state $q_s \in Q_0$ and ends in some state q_f that fulfills an acceptance condition, usually defined as $q_f \in F \subseteq Q$. In general then, and in contrast to deterministic machines, a nondeterministic machine can have multiple successful paths for the same input label σ_{in} resulting in a set of relations $R = \{(\sigma_{in}, \gamma_{out}), (\sigma_{in}, \kappa_{out}) \ldots (\sigma_{in}, \eta_{out})\}$.

A.1.3 State and Machine Equivalence

If a system is described using a finite state machine, it is helpful to be able to decide if some states are equivalent[3]. Equivalent states can be combined, resulting in a lower number of states.

Definition A.1.6. *(State equivalence) Given a machine $M = (\Sigma, \Gamma, Q, q_0, \delta, \omega, F)$ and two states $q_i, q_j \in Q$ then $q_i \equiv q_j$ if and only if $\hat{\omega}(q_i, s) = \hat{\omega}(q_j, s)\ \forall s \in \Sigma^*$*

It can be shown that two states are equivalent if for every input sequence of length $|Q - 1|$, the same output sequence is produced [67]. State equivalence in two distinct machines $M_A = (\Sigma, \Gamma, Q_A, q_A, \delta_A, \omega_A, F_A)$ and $M_B = (\Sigma, \Gamma, Q_B, q_B, \delta_B, \omega_B, F_B)$ can be defined similarly.

Definition A.1.7. *(Containment) Given two machines $M_A = (\Sigma, \Gamma, Q_A, \delta_A, \omega_A, F_A)$ and machine $M_B = (\Sigma, \Gamma, Q_B, \delta_B, \omega_B, F_B)$, M_B contains M_A if for every state $q_A \in Q_A$ there exists at least one equivalent state $q_B \in Q_B$.*

[3] If R is a relation on $S \times S$, then

- R is reflexive if $(a, a) \in R\ \forall\ a \in S$
- R is symmetric if $(b, a) \in R$ then $(a, b) \in R$
- R is transitive if $(a, b) \in R$ and $(b, c) \in R$ then $(a, c) \in R$

An equivalence relation ($\equiv$) is reflexive, symmetric and transitive. Equivalence relations give rise to equivalence classes that contain all and only related members.

Definition A.1.8. *(Machine equivalence) Machine* $M_A = (\Sigma, \Gamma, Q_A, \delta_A, \omega_A, F_A)$ *and machine* $M_B = (\Sigma, \Gamma, Q_B, \delta_B, \omega_B, F_B)$ *are equivalent if and only if machine* M_A *contains* M_B *and* M_B *contains* M_A.

A.1.4 Transformation and Classification

Finite state machines are a model for real systems or behavior, and thus, it helps to understand the nature of this physical behavior, or alternatively, describe which behavior is realizable by finite-state machines.

As is apparent from the above definitions and equations (A.3) and (A.4), a machine transforms an input sequence into an output sequence. Two main characteristics are due to the finiteness of states. For example, it can be shown [67] that a machine with $|Q| = n$ states presented with an input sequence with period p produces an output sequence with period at most np. If the input sequence is composed of indefinite number of identical symbols, then the output sequence will also become periodic. A machine that transforms input sequences into output sequences is sometimes referred to as finite state transducer.

Instead of using a finite-state machine to transform input sequences, it is possible to use it in order to classify those sequences. If for example equation (A.3) is constructed such that given $\Gamma = \{0, 1\}$, for any input $\sigma \in \Sigma$ and any state $p, q \in Q$, the output function is $\omega(p, \sigma) = 1$ whenever $\delta(p, \sigma) = q$ and $\omega(p, \sigma) = 0$ otherwise, then it would be possible to "accept" or "recognize" a sequence that enters the machine into the q state. The same can be achieved - in a more explicit fashion - by using the equation (A.4) instead. The definition of a machine can then be rephrased as:

Definition A.1.9. *(FSA) A finite state automaton is a tuple* $A = (\Sigma, Q, q_0, \delta, F)$ *where* Σ *is a finite input alphabet (or symbols),* Q *is a finite set of states,* $q_0 \in Q$ *is the initial state of the automaton,* δ *is a transition function,* $\delta : Q \times \Sigma \rightarrow Q$ *in the deterministic case or* $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$ *in the nondeterministic case,* $F = \{q_f \mid \forall(p, \sigma),\ \delta(p, \sigma) = q_f \ \wedge\ \omega(p, \sigma) = \gamma\} \subseteq Q$ *is the set of favorable (or final or accepting) states.*

Finite state machines can therefore be defined with respect to their intended domain of application, as automata (recognizers) or transducers.

A.1.5 Machine Description and Properties

The description of a finite state machine is usually given in a transition table or transition graph [68] from which the δ and ω functions can be obtained. For example, the same

machine can be represented by a transition table where rows are a subset of $(Q \times \Sigma \times Q \times \Gamma)$ relations, or a transition graph where edges are labeled with a subset of $(\Sigma \times \Gamma)$ relations as shown in Figure A.1.

Condition		Result	
State i	Input	State $i+1$	Output
q_0	-	q_2	1
q_1	-	q_0	0
q_2	0	q_3	0
q_2	1	q_1	0
q_3	0	q_0	0
q_3	1	q_1	0

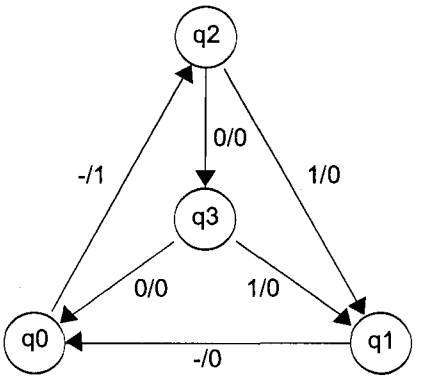

Figure A.1: Representation of a machine using either a transition table or graph

As with any theoretical method that provides a solution to a practical problem, we would like to to able to implement it while minimizing the associated cost. If the description therefore is not in a deterministic form, which it may not be for reasons of easier system characterization, then it needs to be transformed into its deterministic equivalent (see section A.1.3) before being physically implemented, a process called *determinization.* The second concern is with the complexity of the model (both in terms of the number of states and transitions) and thus with the *minimization* of a machine. Since there is no general way to minimize the number of nodes or transitions in a nondeterministic graph [67], minimization is usually applied to deterministic machines.

If we are concerned with classification tasks, there is a straightforward way of transforming a nondeterministic graph into an equivalent deterministic one, as well as minimizing it [81, 69]. In the task of describing input-output transformations, nondeterministic transition graphs pose problems at the determinization level as the equivalence (and intersection) of two nondeterministic transducers are equivalent to the Post Correspondence Problem[4] and are thus undecidable [19].

[4]An instance of the Post Correspondence Problem (PCP)[69] consists of two lists $A = a_1, \ldots, a_k$ and $B = b_1, \ldots, b_k$ of strings over an alphabet Σ and of the question if this instance has a solution. The solution will be a sequence of one or more integers $i_1, \ldots, i_n$ such that $a_{i_1} \ldots a_{i_m} = b_{i_1} \ldots b_{i_m}$

A.1.6 Automata, Languages and Operations

Given a deterministic automaton $A_D = (\Sigma, Q, q_0, \delta, F)$, and using equation (A.1), a language[5] L of A_D denoted $L(A_D)$ [69] is:

$$L(A_D) = \{w \mid \hat{\delta}(q_0, w) \in F\}$$

Similarly, for a nondeterministic automaton $A_N = (\Sigma, Q, q_0, \delta, F)$ where $\hat{\delta}(q, w)$ returns a set of states, a language L of A_N denoted $L(A_N)$ is:

$$L(A_N) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

Note that section A.1.3 defines equivalence of two states in terms of the output function ω. We can restate it in terms of the set F and equation A.1 as follows. Two states $p, q \in Q$ are equivalent if and only if $\forall s \in \Sigma^*$, $\hat{\delta}(p, s) \in F \wedge \hat{\delta}(q, s) \in F$. In general, two automata are equivalent iff $L(A_i) = L(A_j)$, that is, $L(A_i) \subseteq L(A_j)$ and $L(A_j) \subseteq L(A_i)$.

Languages accepted by finite-state automata are called regular languages. Given two languages $L(A_i)$ and $L(A_j)$, then the operations shown in Table A.1 result in some other regular language accepted by automaton A_k [81]. A major result of automata theory is that a language L is accepted by some nondeterministic automaton if and only if L is accepted by some deterministic automaton [69].

If a system is composed of two or more automata executing in parallel, then the state of that system can be expressed in terms of the states of the individual automata. For two automata A_1 and A_2, the system will have $Q_1 \times Q_2$ (or $|Q_1| \cdot |Q_2|$) states. The transition function δ' is a relation on $Q_1 \times Q_2 \times (\Sigma_1 \cup \Sigma_2) \times Q_1 \times Q_2$ and is defined as:

$$\delta'((p, q), \sigma) = \{(p', q') \mid \delta_1(\sigma, p) = p', \delta_2(\sigma, q) = q'\}$$

where $p, p' \in Q_1$, $q, q' \in Q_2$ and $\sigma \in \Sigma_1 \cup \Sigma_2$. If the system halts in some state $(p', q') \in F_1 \times F_2$ then the string is accepted by both automata, which is exactly the result of $L(A_1) \cap L(A_2)$.

A.1.7 Transducers, Relations and Operations

Depending on the problem at hand, finite-state transducers can be interpreted in the following ways [121].

[5] A *regular expression* can also be used to specify all regular languages. Algorithmic procedures exists to convert from a regular expression to a finite state automaton, and vice versa.

Operation	*Representation*
Union	$L(A_k) = L(A_i) \cup L(A_j)$
Complementation	$L(A_k) = \Sigma^* - L(A_j)$
Difference	$L(A_k) = L(A_i) - L(A_j)$
Concatenation	$L(A_k) = L(A_i)L(A_j)$
Kleene star	$L(A_k) = L(A_i)^*$
Intersection	$L(A_k) = L(A_i) \cap L(A_j)$

Table A.1: Operations that result in regular languages

We can view finite-state transducers as finite-state automata with an alphabet $\Sigma = \Sigma_1 \times \Sigma_2$, accepting or rejecting string pairs (a, b) where $a \in \Sigma_1^*$ and $b \in \Sigma_2^*$. This automaton is given by $A = (\Sigma, Q, q_0, \delta, F)$ where $F \subseteq Q$ represents some acceptance condition consistent with the problem at hand, and $\delta = \{(p, (a, b), q) \mid (p, a, b, q) \in Q \times \Sigma_1^* \times \Sigma_2^* \times Q\}$[6].

We can view finite-state transducers as translators when we consider a class of mappings from strings defined on Σ^* to sets of strings $\mathcal{P}(\Gamma^*)$, for example, house $\rightarrow$ $\{maison, casa, Haus\}$. This mapping is rational if it can be realized by some finite-state transducer. Assuming that $\delta(q, \sigma)$ may return a set of states and using equation A.2, we can define a rational transduction $t : \Sigma^* \rightarrow \mathcal{P}(\Gamma^*)$ such that $t(s_i) = \{s_o \mid \exists\, \hat{\omega}(q_0, s_i), s_i \in \Sigma^*\}$ and a rational function if $|t(s_i)| \leq 1, \forall s_i \in \Sigma^*$.

Finally, a finite-state transducer can be viewed as computing relations between sets of strings[7]. By analogy with finite-state automata accepting languages if and only if they are regular (see section A.1.6), transducers accept (compute) relations if and only if they are regular.

Regular relations are defined as [76]: $\{\emptyset\}$ and $\{(a_i, a_j) \mid a_i, a_j \in A\}$. Furthermore, if R_a, R_b and R_c are regular relations, then the following also describe regular relations:

1. $R_a \cdot R_b = \{(a_i b_n, a_j b_m) \mid (a_i, a_j) \in R_a, (b_n, b_m) \in R_b\}$
2. $R_a \cup R_b = \{(r_i, r_j) \mid (r_i, r_j) \in R_a \text{ or } (r_i, r_j) \in R_b\}$
3. $\cup_{k=0}^{\infty} R_c^k = \emptyset \cup R_c \cup R_c \cdot R_c \cup R_c \cdot R_c \cdot R_c \ldots$

[6]This can also be seen as a class of directed graphs where states are vertices and labeled edges are given by $E \subseteq Q \times \Sigma_1^* \times \Sigma_2^* \times Q$

[7]Any subset of the cross-product (including the empty set) between two sets, $A \times B = \{(a, b) \mid a \in A, b \in B\}$, is a binary relation over $A \times B$

Operation	*Representation*
Union	$R(M_k) = R(M_i) \cup R(M_j)$
Complementation	Generally not closed
Difference	Generally not closed
Concatenation	$R(M_k) = R(M_i)R(M_j)$
Kleene star	$R(M_k) = R(M_i)^*$
Intersection	Generally not closed
Cross product	$R(M_k) = L(A_i) \times L(A_j)$
Composition	$R(M_k) = R(M_i) \circ R(M_j)$

Table A.2: Operations that result in regular relations

where R^k are k concatenations of R. By definition, thus, regular relations are closed under the concatenation, union and Kleene-star operations. Operations on regular relations that result in other regular relations are presented in Table A.2. In contrast to the intersection operation on regular languages, regular relations are generally not closed under intersection. To use an oft-cited example, two regular relations $R_1 = (a^n b^*, c^n)$ and $R_2 = (a^* b^n, c^n)$ relate a regular language into another regular language, however $R_1 \cap R_2 = (a^n b^n, c^n)$ relates a non-regular language into a regular language.

If we use the output of one machine as the input into another, the machines are then connected in series (or cascade). It is possible to combine the individual machines into one equivalent machine through the composition operation. Using the definitions for the finite-state machines $M_i = (\Sigma, H, Q_i, \delta_i, \omega_i)$ and $M_j = (H, \Gamma, Q_j, \delta_j, \omega_j)$, then we obtain $M_k = (\Sigma, \Gamma, Q_i \times Q_j, \delta_k, \omega_k)$ where,

$$\begin{aligned} \delta_k((p,q),\sigma) &= \{(p',q') \mid \delta_i(\sigma,p) = p', \delta_j(\omega_i(\sigma,p),q) = q'\} \\ \omega_k((p,q),\sigma) &= \{\sigma' \mid \sigma' = \omega_j(\omega_i(\sigma,p),q)\} \end{aligned} \tag{A.5}$$

The cross-product operation applied to sets of regular languages, $L(A_1) \times L(A_2)$ can be shown [76] to create regular relations. The input strings $L_I(M) = \{s \in \Sigma^* \mid \exists(s,z) \in R(M)\}$ and output strings $L_O(M) = \{z \in \Gamma^* \mid \exists(s,z) \in R(M)\}$ (and thus the input and output automata) can be retrieved from transducer M through projection operations.

A.2 Transducer Properties

Certain transducers can exhibit the following properties. They can be *ϵ-free*, where no edge is labeled with the empty string (the class of transducers with $\Delta \subseteq Q \times \Sigma \times \Gamma \times Q$

is closed under the intersection operation [121]); *minimal*, where no other graph with exactly the same paths that has fewer states exists; *trim*, where all states are accessible from the initial states and where there is at least one successful path from each state (states not meeting this requirement can be removed without changing the capability of the machine); *real-time* where the input label is a single symbol from Σ or $\Sigma \cup \{\epsilon\}$ resulting in $\Delta \subseteq Q \times \Sigma \times \Gamma^* \times Q$.

A.2.1 Unambiguous Transducers

Transducers are *unambiguous* if any input to the machine is the label (see section A.1.2) of at most one successful path. If any input string in Σ^* is mapped to either an empty set or to a set containing only one member, then the mapping represents a rational function (see section A.1.7). Thus, unambiguous transducers realize rational functions, and any rational function π can be represented by an unambiguous transducer if $\pi(\epsilon) = \emptyset \vee \{\epsilon\}$ [121].

A.2.2 Functional Transducers

Any transducer that realizes a rational function is called a *functional* transducer, where functionality is a decidable property of transducers [128]. For example, the class of unambiguous transducers is functional since they realize rational functions. Note that given an input label, if there are two or more successful paths (an ambiguous transducer) but the output labels are exactly the same for all paths, then the transducer still realizes a rational function. In general, the equivalence of two finite-state transducers is an undecidable problem, unless transducers are functional [19]. An example of a functional transducer that is not sequential (and does not have a sequential equivalent [56]) is shown in Figure A.2.

A.2.3 Sequential and p-Subsequential Transducers

A functional transducer that is deterministic with respect to one projection or the other (the automaton that accepts the resulting language is deterministic) is called a *sequential* transducer, and form the most practical class of transducers. By definition, a rational function is sequential if it can be realized by a sequential transducer, and those functions

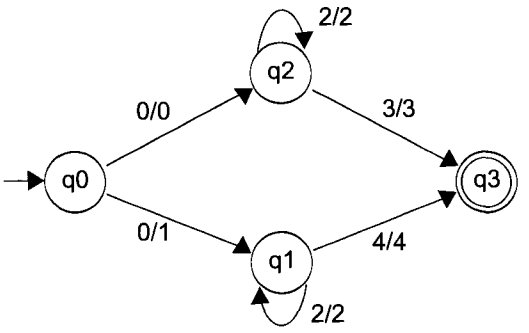

Figure A.2: An example of a non-sequential functional transducer

can be characterized[8] as follows [99]: if f is a rational function mapping Σ^* to Γ^*, then f is sequential if and only if there exists a positive integer K such that $\forall s \in \Sigma^*, \forall \sigma \in \Sigma, \exists w \in \Gamma^*, |w| \leq K : f(s\sigma) = f(s)w$.

Functional transducers can be shown to be exactly the composition of *left-* and *right-*sequential transducers [99]: if f is a function mapping Σ^* to Γ^*, then f is rational if and only if there exists a left-sequential function $g : \Sigma^* \rightarrow \Omega^*$ and a right-sequential function $h : \Omega^* \rightarrow \Gamma^*$ such that $f = g \circ h$, where left- and right-sequentiality refers to the input and output labels respectively.

Sequential transducers are deterministic machines [99] as described in section A.1.1. An equivalent definition is given in [18], where three conditions must be met, namely that the transducer is real-time, that it has a unique initial state, and that for any state and input, there is at most one transition leaving that state and labeled with the input. These conditions ensure that for any input string $s_i \in \Sigma^* \backslash \{\epsilon\}$, there is at most one output string $i_o \in \Gamma^*$ such that (s_i, s_o) is a regular relation. An example of a sequential transducer is shown in Figure A.3 [99].

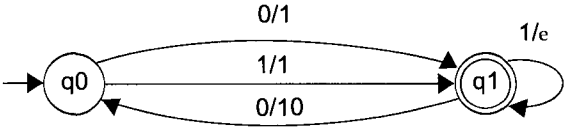

Figure A.3: An example of a sequential transducer

Subsequential transducers are a generalization of sequential transducers, and have additional output symbols based on the final state reached by the transducer during

[8] An example of a function that is not sequential [99] is: $f(w) = \begin{cases} a^{|w|} & \text{if } |w| \text{ is even} \\ b^{|w|} & \text{if } |w| \text{ is odd} \end{cases}$

the computation of the input. Thus, a subsequential transducer is defined as $M_{subseq} = (\Sigma, \Gamma, Q, q_0, \delta, \omega, \rho)$, where ρ is the final emission function, $\rho : Q \rightarrow \Gamma^*$ and other terms are defined as before [121]. For any input string s_i therefore, a subsequential transducers defines a mapping to $\hat{\omega}(q_0, s_i)\rho(\hat{\delta}(q_0, s_i))$ or to $\{\emptyset\}$ if there is no successful path. Subsequential transducers are therefore functional; p-Subsequential transducers on the other hand allow up to $p \geq 1$ final emission strings to be associated with each state, allowing to model a certain finite level of ambiguity in the application. An example of a 2-subsequential transducer is shown in Figure A.4 [99].

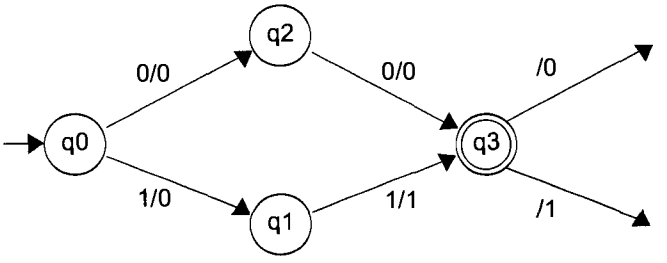

Figure A.4: An example of a 2-subsequential transducer

A.2.4 Classes of Transducers

Finite-state transducers in literature are identified as functional (FUNC), non-functional (NONF), ambiguous (AMBI), unambiguous (UNAM), deterministic (DET), sequential (SEQ), subsequential (SUBS) and p-subsequential (PSUB). The definition of a deterministic machine given in A.1.1 corresponds to the definition of a sequential transducer in [121] and both conform to the definition of an unambiguous transducer, thus UNAM = DET = SEQ. The relationship between those concepts is presented in Figure A.5

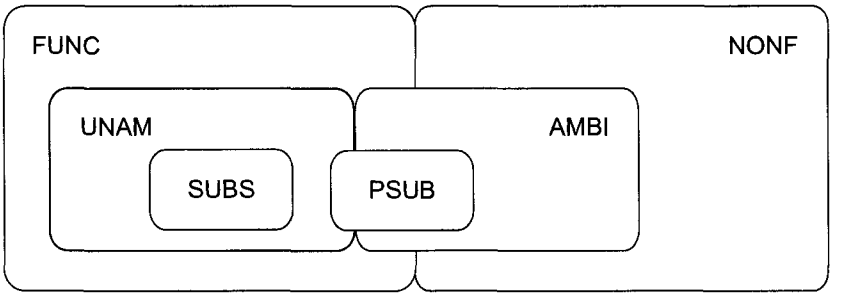

Figure A.5: Relationship between classes of transducers

A.3 Transducer Composition and Decomposition

Decomposition of a machine M with a specified terminal behavior into two or more sub-machines $M_\alpha, \cdots, M_\omega$ can be phrased in terms of a composition problem: find two or more machines which will produce the desired terminal behavior when connected in a prescribed way. This requirement does not imply that the original machine and the combination of sub-machines be equal, or even equivalent. It is sufficient for the combination of sub-machines to contain the original machine. In practical terms it means that the composition can do whatever the original machine can do.

The method of connecting the sub-machines play an important role in both the composition and decomposition problems. Serial and parallel connection methods are the basic building blocks, although more complex connection schemes can be defined. Note that it is assumed that all machines have the same input and output alphabet, and furthermore, in the serial case, $\Sigma = \Gamma$. Parallel composition is associative and commutative and is only defined in case all machines are ϵ-free (see section A.2). The composition operation was described in section A.1.7 where the output and transition functions are given in equation A.5.

In certain cases it is also possible to compose transducers in parallel. The number of resulting states is in practice less than the product of the number of states of the component machines since many states are undefined or unreachable. Using the definitions for the finite-state machines $M_i = (\Sigma, \Gamma, Q_i, \delta_i, \omega_i)$ and $M_j = (\Sigma, \Gamma, Q_j, \delta_j, \omega_j)$, then we obtain $M_k = (\Sigma, \Gamma, Q_i \times Q_j, \delta_k, \omega_k)$ where,

$$\begin{aligned} \delta_k((p,q),\sigma) &= \{(p',q') \mid \delta_i(p,\sigma) = p' \ \wedge \ \delta_j(q,\sigma) = q' \ \wedge \ \exists\, \omega_k((p,q),\sigma)\} \\ \omega_k((p,q),\sigma) &= \{\sigma' \mid \omega_i(p,\sigma) = \sigma' \ \wedge \ \omega_j(q,\sigma) = \sigma'\} \end{aligned} \tag{A.6}$$

Although at first glance, serial and parallel composition of transducers appears very different, in one application area, the finite-state morphology, Karttunen and Beesley [77] write:

> *From a formal point of view there is no substantive difference; a cascade of rewrite rules and a set of parallel two-level constraints are just two different ways to decompose a complex regular relation into a set of simpler relations that are easier to understand and manipulate.*

A.4 Algorithms

Within the class of finite-state machines which are available to describe a variety of problems, Mohri [99] notes:

> *[s]equential transducers are computationally interesting because their use with a given input does not depend on the size of the transducer but only on the size of the input. Since using a sequential transducer with a given input consists of following the only path corresponding to the input string and in writing consecutive output labels along this path, the total computational time is linear in the size of the input [...].*

As such, the following issues are usually addressed: verification that a given transducer defines a functional relation; verification that a given transducer defines a subsequential relation; for transducers that define subsequential relations, determinize it by constructing an equivalent subsequential transducer; and, for subsequential transducers, minimize it by constructing an equivalent minimal subsequential transducer.

A.4.1 Decidability

Schützenberger [128] showed that functionality of finite-state transducers is a decidable property. Choffrut [37] proved that sequentiality of finite-state transducers is decidable and Mohri [99] generalized this to p-subsequential transducers while Roche and Schabes [121] show how to decide subsequentiality of unambiguous transducers. Béal and al. [19] proposed squaring transducers to decide functionality and sequentiality. Gaál [56] proposes an algorithm that decides if a given transducer represents a p-subsequential mapping.

A.4.2 Determinization and Minimization

Determinization algorithms (possibly application-specific) are applied to transducers to obtain sequential or p-subsequential transducers, for example [120, 18].

Minimization algorithms for sequential or p-subsequential transducers reduce the size of the finite-state transducers and examples include [98, 100, 38].

In addition to these algorithms, composition [151], union, equivalence [84] and factorization [80] algorithms for transducers are useful in many instances.

A.5 Illustrative Example

In order to illustrate the concepts introduced in previous sections, consider two finite state machines as shown in figure A.6. The initial state is in bold, accepting states are represented by double circles and $\Sigma \cap \Gamma = \{2, 6\}$. Both machines can be easily verified to be deterministic.

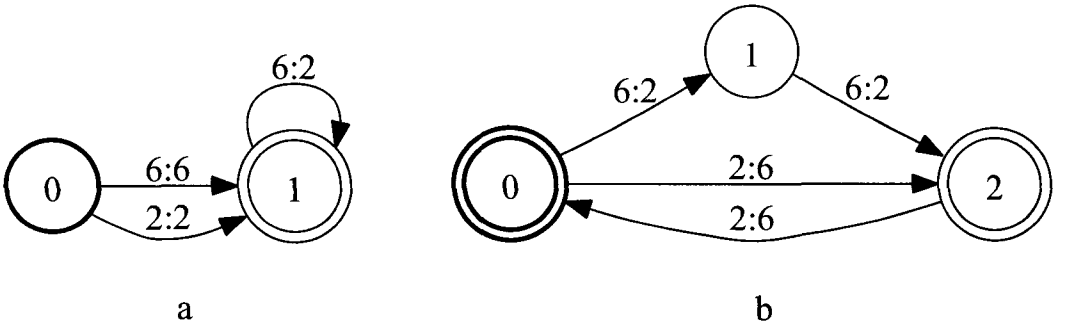

Figure A.6: Component machines

The composition of these machines is presented in figure A.7. Note that inaccessible states (states into which there is no path from the initial state) as well as non-final states out of which there is no transition are not shown. The composed machine thus computes (26*, 66*) relations. This machine is functional (since the component machines are functional) and can easily be verified to be deterministic as well.

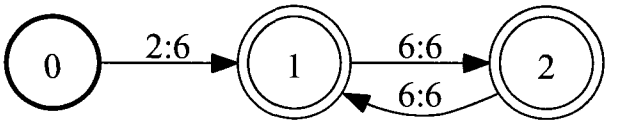

Figure A.7: Composition $(a \circ b)$ of machines in figure A.6

If we are now interested in factorization, that is, given the machine of figure A.7, finding two machines c' and d' such that $c' \circ d'$ gives an equivalent machine, we can use the following procedure. We choose that machine c' will have 2 states $Q_{c'} = \{A, B\}$ where A is the start state and B is the accepting state. Similarly, machine d' will have 2 states $Q_{d'} = \{P, Q\}$ where P is the start state and Q is the accepting state. The composed machine will have $|Q_{c'} \times Q_{d'}| = 4$ states, $Q_{c'd'} = \{AP, AQ, BP, BQ\}$ where AP is the start state and BQ is the accepting state.

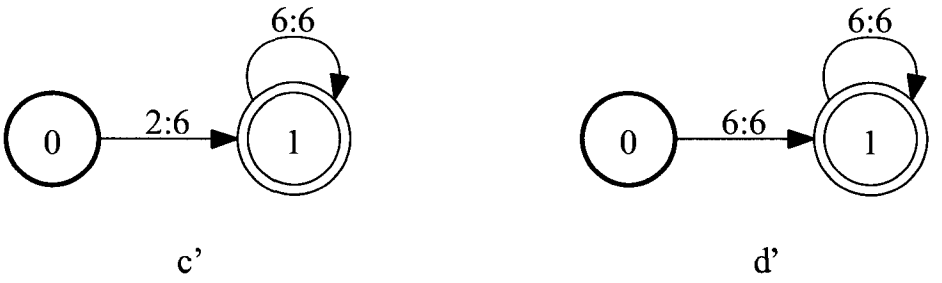

Figure A.8: Factor machines

From the target machine in figure A.7, we can see that from the starting state and on an input symbol "2", a transition occurs to a final state with the output symbol "6". The composed machine must transition from AP to BQ if the input is "2" while issuing an output symbol "6", or using a shorthand notation, $2 : AP \rightarrow BQ : 6$. Therefore the transitions $2 : A \rightarrow B : x$ and $x : P \rightarrow Q : 6$ must be defined for machine c' and d' respectively. The second and final constraint in the target machine is the transition $6 : 6$ from a final state to another final state. This can be reflected in the composed machine as $6 : BQ \rightarrow BQ : 6$ or as $6 : B \rightarrow B : y$ and $y : Q \rightarrow Q : 6$ in machine c' and d' respectively. We can arbitrarily set $x = 6$ and $y = 6$. The transitions are shown in figure A.8.

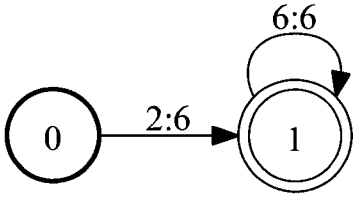

Figure A.9: Composition $(c' \circ d')$ and $(c \circ d)$ of machines in figures A.8 and A.10

The result of the composition of the machines c' and d' is shown in figure A.9. Although this is a very basic example (machine c' is in fact equivalent to the target machine in figure A.7 and machine d' is an identity machine), it illustrates the fact the there are at least two solutions to each factorization.

Note that if we extend the machine c' and d' by $6 : A \rightarrow A : z$ and $z : P \rightarrow R : 2$ where $z = 2$, to obtain machine c and d respectively, the composition $c' \circ d'$ and $c \circ d$ are equivalent. This machines also computes exactly the $(26^*, 66^*)$ relations. Machines

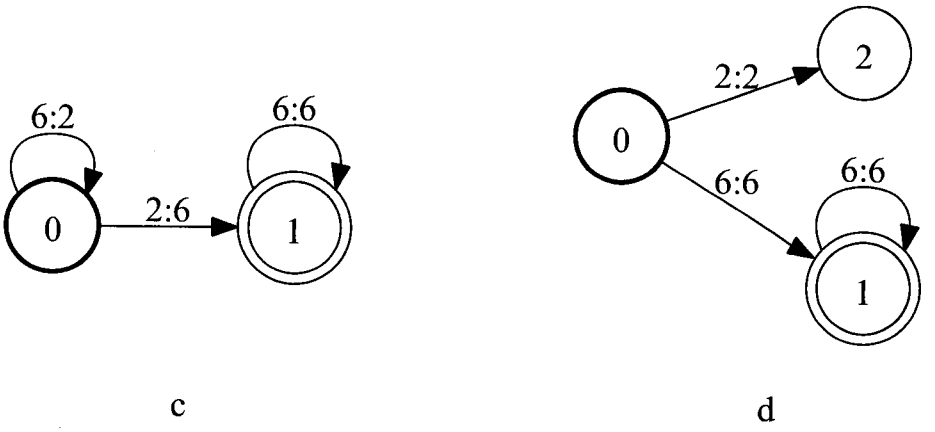

Figure A.10: Extended factor machines

c and d are shown in figure A.10.