Negotiation and Monitoring in Open Environments

Kassidy Patrick CLARK
Technische Universiteit Delft

ISBN 978-94-6203-585-0

Meta-x Meta-write-thesis

[This page intentionally left blank...ish.]

Negotiation and Monitoring in Open Environments

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties
in het openbaar te verdedigen op 14 mei 2014 om 10:00 uur
door

Kassidy Patrick CLARK

Master of Science in de Informatica
geboren te Wiesbaden, Duitsland

Dit proefschrift is goedgekeurd door de promotor:

Prof. dr. F.M.T. Brazier

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof. dr. F.M.T. Brazier	Technische Universiteit Delft, promotor
Dr. M.E. Warnier	Technische Universiteit Delft, copromotor
Prof. dr. A.S. Tanenbaum	Vrije Universiteit Amsterdam
Prof. dr. G. Pierre	Université de Rennes 1
Prof. dr. M.J.G. van Eeten	Technische Universiteit Delft
Prof. dr. J.A. La Poutré	Technische Universiteit Delft
Dr. T.B. Quillinan	Thales Research
Prof. dr. C.M. Jonker	Technische Universiteit Delft, reservelid

Typeset with LaTeX 2$_\varepsilon$
Cover design by Zinnenprikkelend
Printed by CPi Koninklijke Wöhrmann

This research was funded by NLnet:
http://www.nlnet.nl

SIKS Dissertation Series No. 2014-21

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

ISBN 978-94-6203-585-0

Acknowledgements

It was a dark and stormy night in the fall of 2008, when a certain Irishman piqued my curiosity with tales of scientific pursuits. In hushed tones, he described his adventure through the wastelands of decentralized Grid administration to reach a hidden realm, an elite enclave of learned doctors. Pursuing a doctorate is an amazing experience, he told me, a unique opportunity to do science! You are given the chance to devote yourself entirely to a single question, with no limits, no rules and only a little guidance. You get to explore the uncharted jungles of the scientific frontier. To boldly go where no one has bothered to go before. Essentially, you are given a compass and a protractor, but no map. Go! Do science!

Of course, now I know that there are trials and tribulations along the way. Experiments may go wrong or produce unexpected, unexplainable results. You will run into dead-ends and hidden forks in the road. You will make choices and compromises, and you will make mistakes. There will be resistance and conflict, and you may lose some blood along the way. You will also, occasionally, lose your focus and may even succumb to the dreaded *second year slump*. There will be snakes and spiders... and bachelor's students! But, if you stay focused and persevere, you will earn something eternal: the title of doctor.

No one makes this journey alone. You will need a team. You will need heros that have trekked through the same jungle before. You will need jesters to keep your spirits up with care-packages of levity. You will need counselors to talk you off the ledge when an experiment fails, or when you reach a dead-end or a false summit. Choose your team wisely, because they will make all the difference in the world.

My team starts with my promotor, Frances. Without her guidance, encouragement, criticism and, above all else, patience, I would not be the researcher that I am today. She made me better than I was before and, for that, I am grateful.

If Frances was the General, then Martijn was my Captain. Without Martijn, this dissertation would not exist. Full stop. Martijn was my daily supervisor. He kept a close eye on my progress and brandished both the carrot and the stick, when necessary. Martijn was also my mentor, in the fullest meaning of the word. I value his advice and his leadership by example. Not to mention, his mastery of Nethack and Emacs!

Due to my choice of venue, my parents couldn't be with me on this journey, but they laid the foundation for this achievement. They understood the value of education and encouraged me in my own pursuit. From the start of my Bachelor's degree to the finish of my Ph.D. my mother kept my spirits up with encouraging words and the occasional care-package. My grandfather had

only a highschool education. My father raised the bar with two Master's degrees. It hasn't been an easy act to follow, but finally, I am able to raise the bar even farther. For the lessons my parents taught me that prepared me for this journey, I am grateful.

My team also included my in-laws, Rob and Inge. They supported and encouraged me as if I were their own child. In fact, had it not been for a specific conversation with my mother-in-law, I would never have continued studying after receiving my Bachelor's degree. She told me that I would have the rest of my life to work and earn, but I only had these years to study and learn, before my life was filled with distractions and responsibilities. It worked, I earned my Master's and, in doing so, met the group with which I would pursue my Ph.D. For their encouragement, warm food and hours of good conversation, I am grateful.

There were plenty of jesters in my team, including my officemates and paranymphs: Jan-Paul and Michele. While they added absolutely nothing to the content or quality of my dissertation... and sometimes even worked to undermine and devalue my research... and as roommates deliberately created conditions to distract me from my research... and on more than one occasion insulted my intelligence and questioned the very relevance of my research...! Despite all that, I appreciate them for some reason or another. After all, without them, it would have been a pretty dull place.

Anyway, my team also included people who actually contributed something useful to my research. Michel and Reinier helped me with my coding. Thomas helped me with my writing. Many thanks to Vangelis, Yilin, Jordan, Çağri, Sander and all my friends and colleagues for their part in creating a very enjoyable working environment. I also thank Nick for evangelising the Ph.D. life and introducing me to my cube obsession. Of course, I must acknowledge the contribution of the Vrije Universiteit Amsterdam and the Delft University of Technology for providing the incubator for this achievement.

Above and beyond all the other members of my team, there was also one particular, secret ingredient: my wife, Lavinia. She is my rock, my drive, my financier, my biggest fan, my sharpest critic, my guiding light and my inner fire. This achievement is also her achievement. This title is also her title. This day is also her day. For all she has done to make this possible, I am grateful. Thank you.

Kassidy Patrick CLARK
May 2014
Delft

Contents

CHAPTER 1

Introduction

The future brings opportunity. Large scale, distributed, digital environments offer vast potential. Environments that are, in general, dynamic and untrusted. Within these environments, software systems will provide unprecedented support for daily life. Such systems will provide access to vast amounts of knowledge and resources. They will support community-building, resource sharing and enable wider participation of society, at large.

The Smart Energy Grid [61], for example, is being designed to increase sustainability and decrease reliance on fossil fuels. Communities are taking responsibility for their own production. The energy grid is a critical infrastructure on which networked society relies. Intermittent production, however, not only introduces opportunities but also new challenges. Balancing energy load, for example. Producers and consumers negotiate terms and conditions on the basis of which energy is provided and consumed. Service Level Agreements (SLAs) are the result. Conditions may include, for example, penaltes for failure to comply with agreements. Determining that there is a failure requires distributed monitoring.

Society depends on such systems: Systems that rely on technology that is capable of negotiating SLAs and monitoring their success. Multi Agent Systems (MAS) is one such technology. The MAS paradigm offers a straightforward analog for complex systems of autonomous entities. This paradigm is based on the notion of autonomous agents and their interaction. Autonomous software agents represent human actors (i.e. owner) and are capable of negotiating SLAs and coordinating processes with other agents when desirable.

They know their owner's preferences and needs. They are capable of negotiating price, Quality of Service (QoS) characteristics and penalties. They also monitor provisioning of services.

This dissertation presents a MAS framework for distributed negotiation and monitoring of SLAs in large scale, distributed, open environments. This framework enables secure discovery, negotiation and access to distributed resources. Specifically, this dissertation focuses on *open* environments. In this context, an open environment is a large-scale, distributed environment that is also *dynamic* and *untrusted*. The following section elaborates on open environments in the context of this dissertation.

1.1 Open Environments

This dissertation defines open environments as large-scale, distributed, dynamic and untrusted environments. A large-scale, distributed environment connects many users, resources and services across many geographical and administrative domains. A dynamic environment changes over time. Users sporadically enter, interact with and leave an environment. Resource availability and attributes change over time. User requirements and activities change over time. No single authority has complete control over an environment. Users are autonomous and (partially) anonymous, i.e. the identity of a user is hidden from other users. No strict rules govern user actions. An open environment makes no guarantees regarding trustworthiness of users. A user may deceive others about his/her identity or intentions.

An environment may encompass one or more systems. In this dissertation, a system is defined as a (semi) formal construct that connects users and resources. A system defines actions, rules, protocols, permissions, authority and so forth. Operating within an environment, a system must take environmental laws, risks and other influences into consideration. For instance, if an environment is unreliable, a system must provide additional mechanisms to provide reliability. For example, an energy grid is a system that operates within the global weather environment. Changes in weather may affect the system. Therefore, the system includes additional mechanisms (e.g. diversified energy sources, storage, load balancing) to operate reliably.

Open - An open environment is open to users. A user enters an environment, uses resources or interacts with other users at any time. Any users may access an open environment. Multiple authorities may control subsets of components of the environment, but no single authority determines which users are permitted to enter the environment. The environment may grow by

adding new users or connecting to new resources. No single authority controls, monitors or manages environment growth. To communicate and access resources, users require standard rules, syntax and semantics.

Distributed - An open environment is distributed across multiple computers, geographical areas and administrative domains. Large-scale distribution implies thousands or millions of users and resources. Users access resources located in different countries, controlled by different organizations. To operate in such an environment, distributed systems require specialized mechanisms that support user communication, resource discovery and consumption. Distribution also presents challenges, including security and scalability. Section 2.1.1 describes distributed systems and associated challenges in more detail.

Dynamic - An open environment is dynamic and changes over time. Users enter and leave sporadically. User identities are not fixed or permanent. User roles adapt to changing circumstances. In a digital marketplace, for example, sellers become buyers or competitors collaborate. Services and service attributes change (e.g. prices, quality). Service availability (i.e. supply) and consumer demand also change over time. Additionally, the structure of an open environment itself can dynamically change [76]. Open environments offer flexible protocols that enable users to react and adapt to changes.

Untrusted - An open environment is distributed across multiple administrative domains. Therefore, no single actor (e.g. organization or individual) has complete control over an open environment. No single actor regulates all user activity. Users are autonomous and self-interested. A malicious user may lie, cheat and steal for personal gain. Agreements may be violated, rules broken and trust betrayed. In this untrusted environment, protocols must incorporate additional mechanisms to establish and maintain trust between users. Cryptography (see Section 2.1.1), impartial monitoring authorities, audit logs and policies for resolving conflicts between users encourage user trust in the system.

Security and privacy of (sensitive) data is a concern for users. During interactions (e.g. negotiation), users exchange data (e.g. desired price). However, users specifically keep sensitive data secret (e.g. negotiation strategy, highest acceptable price). Mechanisms must support the exchange of certain data while guaranteeing the privacy of other data.

An example of an open environment are future energy markets. In future energy markets, consumers and providers negotiate the sale of energy services. Service availability and price dynamically react to intermittent power generation from green sources (e.g. wind, solar). Consumers dynamically shift

demand in response to market changes. Consumers act to minimize costs and maximize green energy utilization. Providers act to maximize profit and minimize energy waste (i.e. produced but not consumed). Chapter 5 describes future energy markets in more detail.

Open environments present additional challenges to the complex process of service negotiation. Negotiations quickly react to sporadic changes in availability or requirements. Users negotiate despite risks of agreement violation. Design of negotiation protocols in such environments must manage these challenges.

1.2 Research Overview

This section presents an overview of the research presented in this dissertation. Research objectives describe the overarching pursuits of this research. Research questions formalize specific knowledge required to achieve the objectives. An overview of the research approach explains the methods and means applied in this dissertation. Finally, this section summarizes the contributions of this dissertation.

1.2.1 Research Objectives

The overarching objective of this dissertation is to bring the benefits of automated negotiation technology to complex, untrusted environments. The complex nature of such environments impedes manual negotiation and leads to inefficiencies, such as overproduction or unmet demand. Automated negotiation can reduce (human) workload and increase market efficiency. The main objectives of this research are to (1) gain insight in the complexities of negotiation in dynamic, distributed, open environments and (2) provide structure (e.g. languages, protocols, techniques) that supports automated negotiation in such environments.

The following research questions express the knowledge required to achieve the research objectives. The general research question asks:

> **Can a Multi Agent System (MAS) framework be designed to support automated negotiation and monitoring of services in dynamic, distributed, open environments?**

The general research question requires knowledge of negotiation protocols and supporting mechanisms. Two sub-questions, `RQ1` and `RQ2`, highlight specific knowledge requirements for negotiation.

RQ1 Can protocols be designed to support natural negotiation dialogue between agents?

RQ2 Can mechanisms be designed to facilitate reliable, secured negotiation?

The general research question requires knowledge of monitoring mechanisms to support agreement compliance and promote trust. Two sub-questions, RQ3 and RQ4, highlight specific knowledge requirements for monitoring.

RQ3 Can agreements be enforced in a transparent and trustworthy manner?

RQ4 Can trust be established and maintained between agents in untrusted environments?

The remainder of this dissertation pursues knowledge to answer the above research questions. Chapter 3 focuses on RQ1 and RQ2, while Chapter 4 focuses on RQ3 and RQ4.

1.2.2 Research Approach

The *research philosophy* of this dissertation follows the *post-positivist* school of thought [41]. In recent years, post-positivism emerges as an evolution of *positivism* [70]. Positivists gather knowledge of the universe through careful, empirical observation and measurement. The positivist school of thought believes that the universe is comprised of immutable objects that exist independent of an observer. Therefore, empirical knowledge (e.g. measurements, observations) of these objects is objective fact. The values of the observer to not influence the nature of the universe and facts are absolute truth.

Post-positivism builds on the foundations of positivism, but acknowledges that objectivity of knowledge is not guaranteed when the subject matter involves humans [41]. Humans innately "understand" meaning but cannot objectively measure or quantify it [70]. This dissertation presents research on socio-technical systems that involve humans. Therefore, this research follows the post-positivist approach to combine objective measurement with human experience.

The *research strategy* of this dissertation follows the guidelines of *design science* presented by Hevner et al. [68]. Design science focuses on solving a specific (relevant) problem. Research acquires and applies knowledge from theories and measurement to create problem solving artifacts, such as constructs, models, methods and instantiations. Rigorous evaluation of artifacts leads to a cycle of improvement and reevaluation until an artifact adequately solves the given problem. This dissertation presents three main artifacts: (1)

an officially recognized negotiation protocol with extensions, (2) an implementation of a distributed negotiation framework and (3) an implementation of an distributed monitoring framework.

Research instruments enable creation and evaluation of the artifacts [158]. This dissertation uses *literature review*, *experimentation*, *evaluation* and two *use cases*.

Chapter 2 presents a review of literature from related fields and studies. This review provides background knowledge of state of the art solutions and assists evaluation by comparison of the artifacts.

Chapters 3 and 4 design and improve on the negotiation and monitoring artifacts through experimentation and evaluation. Artifacts are implemented and experimentally validated. After experimentation, results are analyzed and evaluated. In addition, the negotiation protocol is evaluated by international peers from an official standards organization.

Chapter 5 presents two use cases that demonstrate the implementation of the research artifacts in two example open environments. Case studies provide qualitative insight into real-world application of the artifacts.

1.2.3 Research Contributions

The main contribution of this dissertation is a framework for service negotiation and monitoring in distributed, dynamic, open environments. The framework consists of several protocols, algorithms and mechanisms to support secure, reliable and enforceable multi agent negotiation. This dissertation describes the design, development, implementation and testing of the framework. This contribution advances the state of the art in automated negotiation and distributed computing. This dissertation also presents both theoretical and practical contributions, including:

`C1` An officially recognized negotiation protocol that supports bidirectional, service negotiation between autonomous software agents. (See `NPS-1` in Section 3.)

`C2` Extensions to the protocol to support symmetrical negotiation roles, dual negotiation states and explicit negotiation semantics. (See `NPS-2` in Section 3.)

`C3` A novel, self-adaptive protocol to securely monitor service agreement compliance in distributed environments and promote trust-building between providers and consumers.

C4 Proof-of-concept implementations of negotiation and monitoring protocols in a distributed, MAS environment offering practical examples to guide future work.

1.3 Dissertation Outline

The structure of this dissertation is as follows:

Chapter 2 - presents an overview of the related fields of research and practice. This overview includes a literature review of the state of art in service negotiation and distributed monitoring.

Chapter 3 - introduces the concepts of service negotiation, service specification and negotiation protocols. This chapter first describes an existing protocol, then a new protocol proposal [C1] with several extensions [C2] and, finally, an implementation and evaluation of the new protocol [C4].

Chapter 4 - introduces the concepts and challenges of service monitoring in distributed environments. This chapter presents several novel monitoring techniques with respective benefits, including a new hybrid approach [C3]. Finally, the chapter describes an implementation and evaluation of the new monitoring technique [C4].

Chapter 5 - presents two examples of distributed, dynamic, open environments. This chapter applies the negotiation and monitoring framework from the previous chapters [C4] to (1) the Smart Energy Grid and (2) Cloud computing.

Chapter 6 - concludes this dissertation with a discussion of the broader implications of this research and areas of future work.

CHAPTER 2

Research Positioning

This dissertation presents a new approach to automated negotiation and distributed monitoring in open environments. The research is positioned at the intersection of the fields of distributed and autonomic computing (see Figure 2.1). This dissertation combines, extends and applies knowledge from these fields to the challenges in open environments, such as trust and dynamism. Automated negotiation and monitoring in such environments requires solutions that are secure, robust, scalable and adaptive. Research in distributed computing offers guidance for designing such solutions. Autonomic computing principles offer insight into designing for adaptability. The autonomic monitoring loop and self-* properties support flexible and autonomous negotiation.

This chapter positions this dissertation within these related fields and presents key concepts, terminology and challenges to which the remainder of this dissertation refers. This chapter also discusses and compares selected related research from these fields, in particular, on negotiation and monitoring. This comparison identifies open issues this dissertation addresses. Finally, this chapter discusses additional technologies that enable application of approaches proposed by this dissertation. These technologies include software agents and the AgentScape middleware.

2.1 Distributed and Autonomic Computing

As discussed in Section 1.1, negotiation in open environments presents several challenges. Principles from the related fields of *Distributed Computing* [39,165]

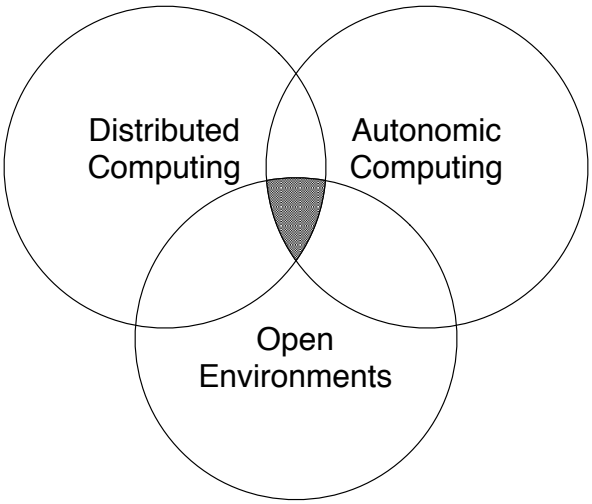

Figure 2.1: Positioning of this dissertation (shaded region) in the related fields.

and *Autonomic Computing* [60, 82, 126] offer insights and solutions to some of these challenges. This section introduces these fields, including key concepts, terminology and challenges.

2.1.1 Distributed Computing

In this dissertation, a distributed computing system is defined as *"a collection of independent computers that appears to its users as a single coherent system"* [165]. Essentially, a distributed system connects multiple users to multiple resources while hiding the complexity of the underlying details. These details include communication, failure, geographical distribution, varying administrative domains and heterogeneity of components (e.g. different operating systems).

An example of distributed computing in the Distributed ASCI Supercomputer [8] - version 4 (DAS-4)[1]. The DAS-4 consists of 198 computer nodes, distributed across 6 locations in the Netherlands. Users are provided an interface that enables concurrent use of one or more nodes.

Another example of distributed computing is the Internet (or more specifically, the World Wide Web). This distributed system enables access to remote resources (e.g. web pages) through a simplified interface (e.g. a web browser). When a user requests a certain remote resource, the request is routed across multiple machines in multiple countries before reaching the target domain. Depending on the size and popularity of a particular web page, the target

[1] `http://www.cs.vu.nl/das4`

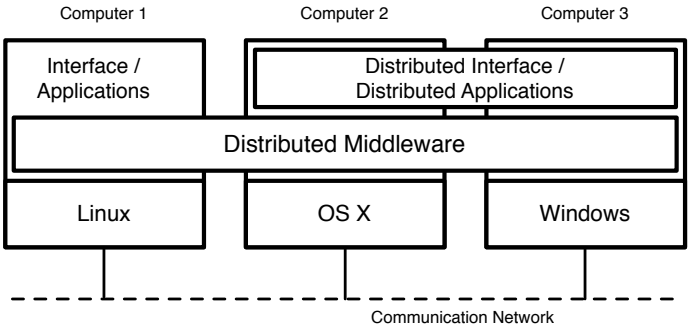

Figure 2.2: A middleware layer distributed across three heterogeneous machines. Adapted from [165].

domain may itself be distributed across several web servers. A web page is returned to the user and displayed in his/her browser. The user's machine, the web servers and the routing machines in between may use different operating systems. The complexity of the heterogeneous machines (e.g. routing protocols, load-balancing mechanisms, fault-tolerant transport layer) is hidden from the user.

Distributed computing systems often use a software layer called *middleware* [165] to bridge the gap between the complex, heterogeneous hardware and a simplified, user interface. Middleware lies between the various operating systems of each machine and the user applications and interface. Figure 2.2 depicts a middleware layer distributed across three heterogeneous machines. Users interact with this layer to access resources from other machines or run distributed applications.

A well-known example of distributed middleware for Grid computing is the Globus Toolkit [58]. The Globus Toolkit includes service libraries, defines protocols and provides a reference implementation for building middleware for Grid computing. Users may use predefined services or build custom applications. A well-defined security model is built into the Globus framework that includes encrypted communication, authentication and nonrepudiation mechanisms.

Another example of distributed middleware is the AgentScape distributed agent middleware. Section 2.3.2 describes this middleware in more detail.

Distributed computing systems are designed using several communication models. Two models relevant to this dissertation are (1) demand-response and (2) message passing. The demand-response architecture forms the basis for many distributed systems, including the World Wide Web. Essentially, all components (e.g. machines, processes) are separated into two groups: clients and servers. A client is active and initiates all communication. In contrast,

a server is reactive and waits for a message to arrive. The communication model is limited to two actions: request and reply [165]. A client requests resources (e.g. services, particular information) and then waits for the reply. A server replies with the requested resource and then waits for a new request to arrive. The static, passive nature of the client-server communication model has several drawbacks. Chapter 3 discusses these drawbacks.

An alternative to the demand-response model is message passing [39]. In contrast to the static roles of client and server, message passing systems enable symmetrical participation. Two applications of message passing are peer-to-peer (p2p) [39] and Multi Agent Systems (MAS) [185]. In these systems, message are passed asynchronously between two ore more autonomous entities, i.e. peers or agents. A peer or agent assumes different roles, depending on a specific relationship with one another. In one relationship, a peer or agent actively initiates communication. In another relationship, a peer or agent responds to incoming requests. As such, each peer or agent supports the capabilities of both client and server.

Designing distributed computing systems is an area of ongoing research [48, 88, 125]. Distributed systems present unique challenges to designers as these systems need to handle several issues, including *heterogeneity*, *security*, *scalability*, *fault tolerance*, *concurrency* and *transparency* [39, 111, 165, 168]. Each of these issues is described below in more detail.

Heterogeneity

A heterogeneous environment is one that contains a degree of variation between components. Each computer may run a different operating system, use a different CPU architecture, support different programming languages and follow different protocols. It is the task of the middleware to accommodate these differences. Middleware can accommodate heterogeneity by running on a variety of architectures and using a "common tongue" to enable different machines to communicate with each other. One specific technology using this approach is a *virtual machine*. Section 5.2 discusses this technology in more detail.

Security

Security often forms a triad of *confidentiality* (i.e. secrecy, privacy), *integrity* and *availability* [105]. Confidentiality of a given resource (e.g. message, data, file) is protected if no unauthorized access is possible. Integrity of a resource is protected if no unauthorized modifications can occur. Availability is protected

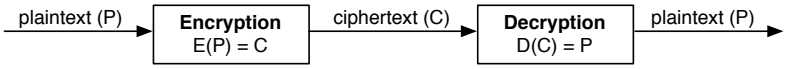

Figure 2.3: The process of encrypting and decrypting a message. Adapted from [154].

if a resource can be accessed, without significant delay, whenever needed. Each of these three aspects is addressed by one or more technologies.

Cryptography is the science of disguising communication to hide its actual content [154]. This includes multiple technologies for *encryption* and *decryption* that are often used to provide security. Figure 2.3 illustrates the processes of encryption and decryption. An unencrypted message is referred to as *plain text*. Once encrypted, the message is referred to as *cipher text*. Encryption ensures that a message or other resource remains confidential.

In addition, certain cryptographic primitives can ensure integrity. One method to ensure the integrity of a given message is through the use of *digital signatures* [154]. A digital signature is a cryptographic "fingerprint" of a given message. A signature is produced by inputting a message into a cryptographic function. Each message produces a unique signature. If a message is modified, the function produces a different signature. Integrity is ensured by comparing signatures to detect modifications.

In addition, a signature also provides *authentication* and *nonrepudiation*. A message is authentic if the sender is (cryptographically) known. A message is nonrepudiable if the sender cannot (cryptographically) deny sending the message. Section 4.2 discusses encryption techniques, including digital signatures and nonrepudiation, in more detail.

Scalability

A system is scalable if it is able to grow and remain functional [39]. More specifically, scalable systems can grow without noticeable effect on performance or administrative complexity [182,183]. This dissertation considers several dimensions of growth, including size, geography and administration [165]. First, system size increases as users or resources increase. For instance, a system can handle one 1 user or 1000 users simultaneously. Secondly, a system can scale across geographic distances. For instance, a system scales to include resources in different regions of a country. Finally, a system can scale across different administrative domains. For instance, a system can scale across different resources controlled by different (private) organizations.

Several issues must be resolved to allow a system to scale. One such issue is centralization of resources [165]. If a particular resource is centralized (e.g. there is only one instance on one server), it may become overloaded

when the number of users increases, creating a bottleneck preventing further scaling. Several techniques are applied to reduce centralization, including distribution or replication of resources (e.g. multiple copies of the instance on multiple servers). Decentralization of resources or algorithms is also used to increase system fault tolerance. For instance, all users share a certain specialized resource (e.g. a print server). If that resource fails, their entire system is affected. However, if the specialized resource is replicated across several locations, a failure of a copy will have limited scope.

The scalability of applications in distributed computing systems can be estimated by measuring the *overhead* generated by applications. A distributed version of a given application generates more overhead than a nondistributed counterpart [135]. Overhead has multiple dimensions, including disk input/output, processing and communication. Additional processing is required for synchronization, security and redundancy. Machines communicate with one another by passing messages across a communications network. The number and sized of messages exchanged partially determines an application's scalability.

Fault Tolerance

A distributed system is reliable if it can tolerate and recover from failures. In distributed systems, it is often difficult to distinguish a slow resource (e.g. overloaded) from a failed resource [165]. Failures may affect only parts of a system and thus be difficult to detect. Special failure detection mechanisms are required.

Redundancy often increases fault tolerance [165]. Essentially, each crucial component has one or more back-up components to take over when the primary component fails. Components for which redundancy is used, include physical components (e.g. 2 power supplies, RAID mirrored hard disks) and software components (e.g. 2 mail applications, 2 processes listening for incoming messages). Components may be large, complex components (e.g. a file server) or small, refined components (e.g. a single process or file). Redundancy ensures that data is *persistent*. Data is preserved and is not lost or corrupted during failures.

Steps can be taken to mitigate or entirely hide failures from users or applications [39]. For instance, if a message is lost in transit, it may be automatically retransmitted without notifying the user.

Concurrency

Multiple users access shared resources at the same time. Concurrent access creates complex state transitions for these resources. If transitions occur in an

unintended order, the result may be invalid. For instance, if two users access a shared integer (e.g. 10). One user decrements the integer. One user doubles the integer. Depending on the order of the operations, the value of the integer is either 18 (i.e. $10 - 1 = 9, 9 * 2 = 18$) or 19 (i.e. $10 * 2 = 20, 20 - 1 = 19$). Concurrent access may also result in lost operations (i.e. operations whose effects are undone or overwritten). For instance, if the first user reads the current value of the integer, then decrements the value. However, before the first user can save the new value to the integer, the second user reads, doubles and saves a new value (i.e. 20) to the integer. Finally, the first user saves the new value (i.e. 9) to the integer. The value of the integer reflects only the operations of the first user and the actions of the second user are overwritten and lost.

Race conditions arise when operations occur in an unintended order [113]. The challenge of concurrency is to ensure that resources remain in a valid state. Mechanisms prevent lost operations by ensuring *mutual exclusion* to such resources. Mutual exclusion mechanisms synchronize access to critical resources. Note that synchronization in distributed systems presents a unique challenge as no global clock can be assumed [39]. Two machines may disagree on the exact time that a given message arrives. Therefore, additional algorithms are required to determine the correct order of messages.

Transparency

Distributed computing systems require mechanisms to handle the complexities of heterogeneity, security, scalability, communication, fault tolerance, and concurrency. In the example of the World Wide Web as a distributed system, the complex inner workings of distributed systems are often invisible to the user. A *transparent* system hides this inner complexity [165]. The end user experiences a single, coherent machine. The level of transparency influences other requirements, such as customizability, performance and usability [168].

2.1.2 Autonomic Computing

Computing systems are becoming more complex. Systems are becoming larger, more heterogeneous and dynamic. At the same time, systems are becoming more critical to today's modern society. Large, complex systems underpin economical infrastructure and daily life. For instance, complex systems control vast numbers of banking transactions, communication and logistics. These

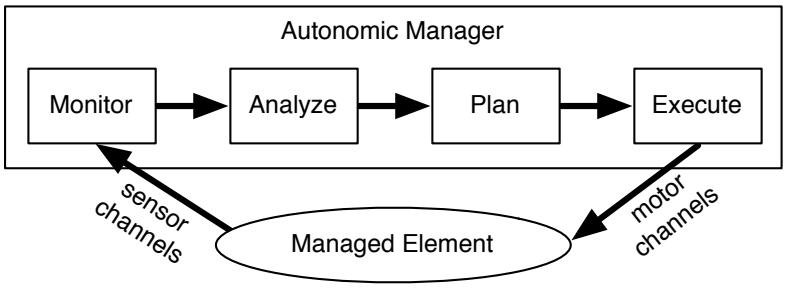

Figure 2.4: Common architectural approach to building autonomic element. Adapted from [82].

systems must be installed, configured, maintained and upgraded. The complexity of these management activities is reaching the limits of human administrators. In response, the field of autonomic computing emerges to create systems capable of self-management [60, 82, 126].

Autonomic principles apply to both low level components (e.g. a single process, a hard disk), collections of components (e.g. an application, a machine) and large scale systems, such as a distributed application or computer. Each autonomic element requires (1) sensor channels to detect changes and (2) motor channels to react to these changes [126].

Figure 2.4 illustrates a common architectural approach to building autonomic elements [82]. An autonomic manager is responsible for each element (e.g. hardware resource). The manager consists of four key processes: *monitor*, *analyze*, *plan* and *execute* (MAPE). Via sensor channels, the manager collects monitors data. This data is analyzed to detect changes (e.g. failures, increased CPU load). Corrective action is planned (e.g. rebooting, modifying a particular variable). Via motor channels, the manager executes the plan.

An autonomic system is characterized by several properties, referred to as self-* properties. These properties include *self-awareness*, *self-configuration*, *self-optimization*, *self-healing* and *self-protection* [60, 126].

An example web server illustrates these properties. A given organization operates an internal web server that hosts a page showing a set of dynamically computed statistics. These statistics reflect the current number of users logged on to the organization's network, the average users per hour, the average duration of a user's session and so forth. Each time the page is requested, the server recomputes the statistics. This process involves retrieving several values from one or more (remote) databases, calculating the averages and generating the web page. Depending on the number of requests per second, this process can generate significant network traffic and CPU load.

Self-awareness

A self-aware system is explicitly aware of itself and its environment. This includes knowledge of internal state, behaviors, resources, policies and abilities. A systems monitors key metrics, such as resource usage, performance statistics and environmental variables. In the above example of a web server, the server monitors network traffic and CPU load. Self-awareness also reflects knowledge of possible actions. For instance, knowing which actions are possible and what effect they have. Self-awareness is fundamental to the remaining self-* properties.

Self-configuration

A self-configuring system automatically adapts to changes in the environment by reconfiguring itself. In the case of the web server, the server responds to high CPU load by caching the statistics page. This reduces CPU load by giving users a static copy of the page rather than recomputing it for each request.

Self-optimization

A self-optimizing system monitors itself and fine tunes various settings to maximize a given goal. The web server from the example above fine tunes the caching settings to maximize requests per second and minimize the age of a given statistic. For instance, with 1000 requests per second, the statistics are computed every 10 seconds. Therefore, a user may receive statistics that are 10 seconds outdated. As requests per second increases, the interval between statistical computations gradually increases (e.g. 1 minute, 10 minutes).

Self-healing

A self-healing system automatically discovers, diagnoses and recovers from failures. Failures occur at many levels, from a failed process to failed hardware. A system detects disruptions, discovers the relevant process or component, diagnoses the cause of failure and attempts recovery (e.g. restarting a process or rebooting a machine). A self-healing web server automatically detects a failure in a web service, determines and restarts the responsible module.

Self-protection

A self-protecting system automatically detects, identifies and defends itself from attack. Monitoring detects unauthorized attempts to access or modify

resources. The example web server monitors login attempts. If suspicious activity is detected (e.g. 100 failed attempts within 10 seconds), the offending account (or IP address) is automatically blocked.

2.1.3 Open Environments

This dissertation defines open environments as large-scale, distributed, dynamic and untrusted environments. No single authority controls the entire environment or governs the actions of all users. Users are autonomous and (partially) anonymous. A user's identity or location may be hidden. An open environment does not specify or prevent "unacceptable" user actions. Therefore, a user may deceive others about his/her identity or intentions. Lack of trust between users requires additional security considerations when operating in such environments (e.g. negotiating with other users). Open environments require additional mechanisms to provide privacy of (sensitive) user data, manage risk and offer assurance. Section 1.1 above defines open environments in more detail.

2.2 Related Work

This section provides an overview of research related to the contribution of this dissertation. Related research is compared and categorized based on the criteria of open environments. This research is related to two specific research fields: negotiation and monitoring.

2.2.1 Negotiation Research

This section compares automated, distributed negotiation protocols for use in open environments. The comparison uses two main criteria: *symmetry* and *multiround* support. In this dissertation, a symmetric protocol enables all negotiation participants[2] with equal access, action, privilege and responsibility, regardless of role or function. Symmetry of roles (i.e. the consumer and provider are equals) allows for flexible protocols in open environments. In such environments, roles are flexible and ambiguous. A "consumer" may resell an object of negotiation, thus simultaneously becoming a "provider". A change of role should not require a change in the underlying protocol. For instance, if a consumer wishes to become a provider, this change should not require additional libraries or request of additional methods or permissions. The change

[2]When engaged in negotiation, users are referred to as *participants* of a negotiation process. Each participant may assume a specific role during negotiation, such as *consumer* or *provider*.

of roles should be fluid and instantaneous. Each role should have equal (i.e. symmetric) abilities (e.g. initiate negotiation) and equal access to negotiation data. The concept of role symmetry is not new. In fact, it is one of the design conventions of automated negotiation proposed by Rosenschein and Zlotkin in 1994 [147]. However, many protocols do not include symmetry as a design goal.

Another criteria by which negotiation protocols are compared is whether the protocol supports multiple rounds of negotiation. A single round of negotiation consists of a single request and a single response. For instance, a consumer proposes a price and a provider accepts or rejects it. Within this model of interaction, there is only a single chance to reach agreement. If a price is rejected, a negotiation is completed without reaching successful agreement. In contrast, multiple rounds of negotiation allow participants to explore possibilities, improve offers and increase the chance of reaching agreement. For instance, a consumer suggests a price, a provider rejects the price, the consumer increases the price and the provider accepts the higher price. This model of interaction is called *multiround* negotiation. The dynamic nature of open environments leads to changing services, requirements and attributes (e.g. a lower price, a higher quality). In dynamic environments, multiple rounds of negotiation provide agents the possibility to together search through these changes to find a mutually acceptable agreement.

Figure 2.5 positions automated, distributed negotiation protocols found the literature in one of four quadrants, based on the above mentioned criteria. `Q1` contains monitors that are both symmetric and support multiround negotiation. `Q2` contains asymmetric protocols that support multiround negotiation. `Q3` contains asymmetric protocols that do not support multiround negotiation. `Q4` contains symmetric protocols that do not support multiround negotiation.

`Q1` - This quadrant contains distributed negotiation protocols well-suited to open environments. A protocol in this quadrant is designed with symmetric negotiation roles. A consumer or a provider have equal abilities and equal access to negotiation data. Both may change roles without requiring a change of protocols, abilities, permissions or data.

Additionally, a protocol in this quadrant supports multiple rounds of negotiation. The negotiation process is a bidirectional dialogue. For instance, a consumer proposes a price, a provider proposes a much higher price, the consumer proposes a slightly lower price and so on. Chapter 3 presents a negotiation protocol designed for this quadrant.

`Q2` - This quadrant contains asymmetric negotiation protocols that support multiple rounds of negotiation. The Iterated Contract Net Interaction

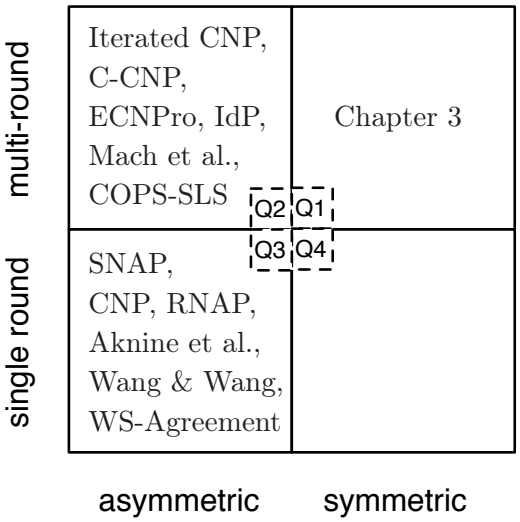

Figure 2.5: Comparison of related negotiation research.

Protocol (Iterated CNP) is a FIPA[3] standard that extends the Contract Net Protocol (CNP) (see Q3) with multiple rounds [57]. A CNP negotiation round consists of a consumer creating a request and one or more providers submitting offers. The iterated variant of CNP allows the consumer to repeat this process with slightly modified requests to guide the resulting offers (e.g. proposing a lower price). Vokřínek et al. present Competitive CNP (C-CNP) that extends Iterated CNP with additional explicit phases for decommitment and contract termination, enforced with (monetary) penalties [171]. Another extension to Iterated CNP is ECNPro presented by Wong and Fang [184]. ECNPro supports multilateral (i.e. one-to-many) negotiation between a single consumer and multiple providers. Consumer requests can be divided into sub-requests and concurrently negotiated with multiple providers. These extensions follow the rules of Iterated CNP and thus support multiple rounds of negotiation but lack symmetric roles.

Mach et al. propose a bilateral bargaining protocol in [100]. A high-level overview of their negotiation pattern indicates support for multiple rounds of negotiation (e.g. offer, counter-offer). The consumer and providers roles appear symmetrical, except that only the consumer is able to accept or reject an offer. The provider may only propose counter-offers or create agreements

[3]Foundation for Intelligent Physical Agents, http://www.fipa.org/

in response to offers. At the time of writing, this protocol is not implemented, thus closer study is not possible.

Green et al. present the Intra-domain Protocol (IdP) for use in the Quality of experience Delivery In New generation telecommunication networks with E-negotiation (QDINE) negotiation framework [63]. The framework identifies 5 distinct roles, including consumer, provider and market agent (i.e. mediator or broker). IdP supports direct negotiation between consumer and provider or mediated negotiation through a market agent. IdP also supports negotiation with multiple providers simultaneously. This negotiation protocol proceeds as follows: (1) a consumer requests a service; (2) a provider accepts, rejects or proposes a counter-offer; (3) the consumer accepts, rejects or proposes a new counter-offer; (4) after one or more rounds of proposals, an agreement is created or the session terminates.

Nguyen et al. propose the Common Open Policy Service protocol for Service Level Specification (COPS-SLS) [114]. COPS-SLS extends the general purpose COPS protocol [18] for negotiation of network level SLAs. Policy Decision Points (i.e. provider) and Policy Enforcement Points (i.e. consumer) negotiate quality of service for network services, such as bandwidth. Communication is initiated by the consumer and occurs in two phases: configuration and negotiation. During configuration, a consumer and provider discover the negotiation context, such as the maximum lifetime of an agreement and maximum values (e.g. maximum bandwidth available). The negotiation phase proceeds as follows: (1) a consumer requests (REQ) a service configuration; (2) a provider decides (DEC) to accept, reject or propose an alternative configuration (e.g. counter-offer); (3) the consumer either reports (RPT) if the offer is accepted or rejected, or the consumer requests (REQ) an alternative configuration (e.g. counter-offer). The protocol supports multiple iterations of requests and decisions. In addition to these steps, a provider may send an unsolicited decision (DEC) to degrade the service, if necessary.

Q3 – This quadrant contains asymmetric negotiation protocols that do not support multiple round of negotiation. The Service Negotiation and Acquisition Protocol (SNAP) offers a high-level overview of operations for SLA creation in distributed environments [43]. A clear distinction is made between clients and resource owners. These roles have different actions and access. The issue of multiround negotiation is less clear. The authors stress the importance of multiphase negotiation as a tool to explore the negotiation space. However, SNAP operations do not explicitly support multiround negotiation.

Another protocol for establishing agreements in distributed environments is the Contract Net Protocol (CNP) defined by Smith and Davis [151, 159].

Two distinct roles are the manager (e.g. consumer) and contractor (e.g. provider). These roles have different abilities, operations and permissions. For instance, negotiation is always initiated by a consumer. Providers submit offers from which the consumer chooses the most acceptable. The protocol terminates after this single round. Several extensions to CNP add features, such as concurrent negotiation sessions proposed by Aknine et al. [1] or a bulletin board (publish-subscribe) communication model proposed by Wang and Wang [175]. These extensions follow the rules of CNP and thus lack support for multiple rounds of negotiation or symmetric roles.

The Web Service Agreement (WS-Agreement) specification defines a protocol for SLA creation [5]. Different operations are defined for consumer and provider roles. Interaction is limited to a single round. A consumer makes an offer and the provider accepts or rejects it. Section 3.2 describes WS-Agreement in more detail.

Wang and Schulzrinne introduce the Resource Negotiation Protocol (RNAP) in [176] for resource allocation in distributed environments. The protocol supports message exchange between a Host Resource Negotiator (HRN) (i.e. consumer) and a Network Resource Negotiator (NRN) (i.e. provider). The message exchange is always initiated by the HRN and comprise the following steps: (1) a consumer sends a *query* requesting current resource prices; (2) a provider responds with a *quotation* containing pairs of services and current prices; (3) the consumer chooses one or more services with a *reserve* message; (4) the provider responds with a *commit* message stating that the reservation is either accepted or rejected; (5) after service consumption, the consumer sends a *close* message, and (6) the provider ends the service with a *release* message. These steps constitute a single negotiation round. If negotiation does not result in a successful agreement, the consumer has no option to suggest an acceptable price (e.g. counter-offer).

Q4 - This quadrant contains symmetric negotiation protocols that do not support multiple rounds of negotiation. The literature review does not include service negotiation protocols that incorporate symmetry as a design goal. Rather than defining one set of actions for all negotiation participants, regardless of role, the reviewed protocols assign different actions, privileges and responsibilities to different roles (e.g. consumer, provider). To create symmetric roles, ad hoc solutions assign multiple roles to each participant. For example, WS-Agreement allows each participant to offer instances of both the client and server Application Programming Interfaces (API).

2.2.2 Monitoring Research

This section compares research on monitoring based on suitability to open environments. This comparison uses two main criteria: *trust* and *adaptation*. Trust considers whether a monitor is designed for trusted or untrusted environments. Open environments make no guarantees that users are trustworthy. Therefore, monitors in these environments are designed to protect against malicious, deceitful users. *Is the monitoring process transparent to all participants? Can monitoring results be accessed and audited by all participants?*

Adaptation considers whether a monitor adapts to changes, such as environmental or policy changes. *Can the monitor adapt itself to increased load or a user's changing requirements?* Changing circumstances in dynamic, open environments require adaptive solutions.

The selected monitoring research is limited to distributed, service monitors. These monitors are designed for distributed environments and thus address the issues of distributed computing discussed in Section 2.1.1, including scalability.

Figure 2.6 positions monitoring research found in the literature in one of four quadrants, based on the above mentioned criteria. `Q1` contains monitors that are both adaptive and suited to untrusted environments. `Q2` contains adaptive monitors designed for closed, trusted environments. `Q3` contains monitors that cannot adapt to changing requirements and operate only in trusted environments. `Q4` contains monitors that cannot adapt, but are designed for use in open, untrusted environments.

`Q1` – This quadrant contains monitors well-suited to open environments. A monitor in this quadrant adapts to the environment. It senses changes in requirements or priorities and adjusts itself accordingly. A monitor in this quadrant is designed for use in an untrusted environment. It includes processes and mechanisms to guarantee objectivity of monitoring results. Mechanisms prevent unauthorized, malicious modification of monitoring data. All participants can access, audit and verify monitoring data. Chapter 4 presents a monitor designed for this quadrant.

`Q2` – This quadrant contains monitors able to adapt to changes, but not suited to untrusted environments. Some monitoring frameworks are able to dynamically adapt to changes in an environment or (internal) policy. Keung et al. propose a self-adaptive, self-optimizing extension of the Monitoring and Discovery System (MDS3), based on Globus Toolkit [83]. The monitoring approach proposed for MDS3 collects measurements from distributed nodes. The frequency of these measurements is dynamically adjusted in response to changes in CPU load. For instance, higher load (e.g. more users in the system), the lower the frequency of measurements.

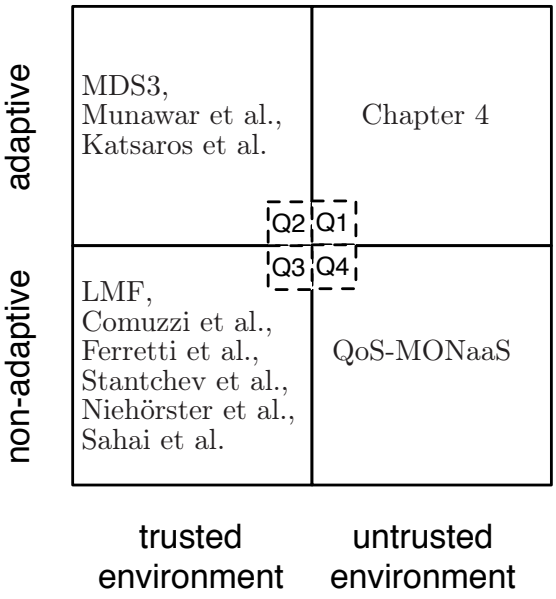

Figure 2.6: Comparison of related monitoring research.

Munawar et al. describe another example of an adaptive monitor in [110]. This monitor reduces processing overhead by preselecting key metrics. During normal operation, only these metrics are monitored. If an anomaly is detected, the monitor adapts by increasing the number of related metrics that are monitored. The increased number of monitored metrics offers higher monitoring assurance that failures are detected at the cost of higher processing overhead.

Katsaros et al. present a self-adaptive, hierarchical monitoring mechanism for Cloud environments [80]. The monitor is distributed across the Software-, Infrastructure- and Platform-as-a-Service layers. The Software layer allows users (i.e. Cloud consumers) to specify Key Performance Indicators (KPI), choose monitoring metrics (i.e. the list of measured resources) and adjust measurement intervals at run time, based on the requirements of a specific application. The Infrastructure layer provides access to low-level metrics (e.g. CPU, network latency). The Platform layer analyzes monitoring results and takes corrective action if necessary. The consumer subscribes to the monitoring service to receive periodic notifications. While the consumer is able to specify and adjust metrics and intervals during runtime, the data collection and storage is controlled by the Cloud Service Provider (CSP). The consumer implicitly trusts that measurement data is not modified or deleted by the CSP.

Q3 – This quadrant contains monitors unable to adapt to changes and not suited to untrusted environments. The Lattice Monitoring Framework

(LMF) monitors resources in virtualized, distributed environments, such as the Cloud [35]. This framework is designed for use in a closed, trusted environment. LMF is designed from the perspective of the provider and is completely hidden from the consumer. The resource (e.g. Cloud) provider has complete control over all aspects of the monitor. As such, the consumer cannot access or verify monitoring results. This monitor is not suited to untrusted environments in which a provider may deceive a consumer.

Comuzzi et al. present a monitoring framework that also focuses on monitoring from the provider's perspective [37]. Monitoring data is collected and stored by the provider. No mechanisms ensure objectivity or integrity of monitoring results. Consumers must trust the provider not to maliciously modify results.

Ferretti et al. and Stantchev et al. present approaches for monitoring Quality of Service (QoS) in the Cloud [54,160]. Providers monitor certain metrics to prevent over- or under provisioning of resources. In effect, these monitors act as load-balancing mechanisms. These monitors are controlled by the resource provider and transparency of the monitoring process is not considered.

Niehörster et al. present a mechanism for enforcing service agreements for Grid computing is in [115]. This monitor uses software agents (see Section 2.3.1) to monitor compute jobs: one agent per job. An agent assigns resources to ensure that a job is completed in accordance with the agreement. The monitoring process is controlled by the provider. The consumer has no ability to verify agreement compliance. As with the other monitors introduced in this section, consumers must implicitly trust providers. However, no mechanisms are in place to prevent deceitful providers from secretly modifying monitoring data.

Sahai et al. present an architecture for specifying and monitoring SLAs in commercial Grids [149, 150]. Monitoring data is collected from relevant locations (e.g. provider components, consumer applications) and stored a central repository for compliance analysis. The monitor cannot adapt to changes in requirements. A commercial Grid is a controlled environment that assumes trust between users.

Q4 – This quadrant contains monitors unable to adapt to changes, but suited to untrusted environments. In contrast to the monitoring approaches introduced above, Romano et al. introduce the QoS-MONaaS framework that acknowledges and addresses the issue of trust [145]. This framework incorporates an anonymizing function that protects objectivity of monitoring results. This function anonymizes requests made to the monitor. The monitor cannot determine if a request is made by a provider or a consumer. As such, there is no incentive to tamper with monitoring results. For instance, if the monitor

wishes to deceive the consumer about actual performance measurements. The monitor is unable to supply the consumer with false results, while supplying the provider with accurate results.

2.3 Enabling Technologies

This section introduces two technologies that enable the application of the distributed and autonomic principles discussed above. These technologies include software agents and the AgentScape middleware. The Multi Agent System (MAS) paradigm enables autonomous action (e.g. problem solving, decision making) and social interaction (e.g. communication, negotiation) [22, 74, 185]. The AgentScape middleware enables distributed, multi agent applications. AgentScape includes mechanisms for security and reliability. Together, AgentScape and software agents enable autonomous activities in distributed, dynamic, open environments.

2.3.1 Software Agents

Jennings et al. define agents as software that is capable of flexible, autonomous actions that allows an agent to adapt to given circumstances [75, 76]. Flexible autonomy is characterized by agents that are (1) responsive, (2) proactive and (3) social. *Responsive* agents are aware of and react to environmental changes. *Proactive* agents predict changes in the environment or situation and take preemptive action. *Social* agents interact with other actors (e.g. humans, other agents) in the environment to solve problems or achieve goals. Social interaction between agents occurs through the passing of messages.

Software agents commonly automate activities, such as negotiation [11, 13, 21, 25, 76, 78, 85, 87, 136, 139, 152, 153, 173]. In such scenarios, software agents represent participants (e.g. consumer, provider) in the process of negotiation. Agents encapsulate decision-making strategies and are able to act autonomously to achieve a desired goal.

As stated in Section 1.1, open environments present challenges to negotiation. Such environments are distributed, untrusted and highly dynamic. The structure of the environment, resources, participants, requirements and strategies change over time. Agents are well-suited to such open and complex environments [76].

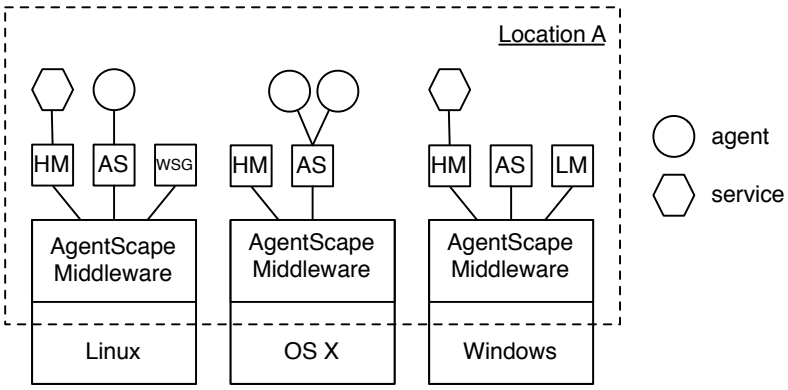

Figure 2.7: AgentScape distributed middleware.

2.3.2 AgentScape Middleware

AgentScape[4] is a distributed middleware framework that supports scalable, secure, open, fault tolerant, heterogeneous, mobile, Multi Agent Systems (MAS) [121]. The AgentScape middleware enables rapid prototyping and deployment of MAS services in open environments. The technologies presented in this dissertation are experimentally validated with this middleware.

Figure 2.7 depicts the conceptual structure of this middleware. An *AgentScape Location* is an administrative domain that groups one or more machines together. A Location may comprise several geographically distributed machines running different operating systems (e.g. Linux, OS X, Windows). Each Location has a single *Location Manager* (LM) responsible for regulating access to a Location and resources. Resources include *Agent Servers* (AS) that host mobile agents for different programming languages (e.g. Java, C, Jason). Each separate machine has a *Host Manager* (HM) responsible for regulating access to a host and resources. A *Web Service Gateway* (WSG) provides access to external web services. A collection of Locations that are aware of, and accessible to one another, is referred to as an *AgentScape World*. An external *Lookup Service* is responsible for providing listings of known locations and services.

AgentScape supports agent migration between machines and locations. For instance, an agent is created on a Linux machine. The agent searches the LS for a desired service on a Windows machine. Once found, the agent requests access to this service by contacting the HM of that particular machine. Once access is granted, the agent migrates to that machine and consumes the service. Migration between Locations works similarly. Agent migration enables

[4]More information, including source code available at: `http://www.agentscape.org`

agents to offer and consume resources distributed geographical or administrative domains.

2.4 Conclusion

This dissertation presents research on designing a framework for automated negotiation and distributed monitoring in open environments. The research draws on the related fields of distributed and autonomic computing. This chapter positions this dissertation within these fields. The following chapters refer to the principles and terminology of distributed and autonomic computing.

Within this context, this chapter compares and categorizes related research. Some related negotiation research supports multiround negotiation, but not symmetry of roles. Chapter 3 presents a multiround, symmetric negotiation protocol. Several approaches to distributed monitoring are dynamic and able to adapt to changes in the environment. Other approaches are suited to untrusted environments. Chapter 4 presents an approach that is both adaptive and suited to untrusted environments.

CHAPTER 3

Service Negotiation in Open Environments*

Negotiation is a bidirectional dialogue between two or more users, with possibly conflicting goals, that together search for a mutually acceptable agreement [75]. When engaged in negotiation, these users are referred to as *participants* of a negotiation process. Each participant may assume a specific role during negotiation, such as *consumer* and *provider*. Consumers and providers often negotiate access to resources and services. In energy marketplaces, for example, consumers negotiate energy services with providers. During negotiation, participants often exchange messages (e.g. offers, counter-offers). If successful, negotiation results in an agreement that specifies the terms and conditions of the service.

Negotiation is often a complex process as participants may pursue conflicting goals. For instance, a provider often attempts to maximize the price of a service, whereas a consumer often attempts to minimize this price. Negotiation participants follow negotiation strategies to achieve negotiation goals. For example, a provider may first decide upon a minimum price for a given service and then initiate negotiation (e.g. first offer) with a price far above this minimum. The provider predicts that the negotiation process will ultimately lead to a compromise below this initial price, but still above the chosen minimum. Negotiation strategies may also change during negotiation to adapt to new

*This chapter is based on three published papers [12, 28, 174].

information or requirements. The field of game theory [14,146] studies negotiation strategies in depth (e.g. reaching a Nash Equilibrium or Pareto-efficient outcome).

The complexity of negotiation limits the speed and efficiency of human actors. To address these issues, (semi) autonomous, software agents (see Section 2.3.1) are used to automate the negotiation process. Agents represent (human) participants in the negotiation process. Agents act on behalf of respective participants in accordance with defined preferences and goals. This dissertation presents a Multi Agent Systems (MAS) approach in which a reference to a particular agent is a reference to the (human) participant that agent represents.

The field of automated negotiation covers many issues, from defining negotiation strategies to designing supporting frameworks. While other researchers focus on the former [69, 92, 93, 139], this dissertation focusses on the design of a framework to support automated negotiation in open, distributed environments. Multiagent, automated negotiation in requires well-defined structures, such as *specification languages* and *negotiation protocols*. An unambiguous language specifies the resources and services to be negotiated. This includes aspects of the services, such as names, locations, prices, sizes, amounts and durations.

A negotiation protocol defines a frame of reference for negotiating agents. A protocol defines how agents communicate, what terminology they use and what actions they are able to perform. A protocol may also determine the order of events; for example, which participant is allowed to initiate negotiation. A protocol limits and controls the type of information (agent) participants exchange. This determines whether negotiation involves a single issue (i.e. a single service) or multiple issues (i.e. complex services). The design of a negotiation protocol fundamentally influences the subsequent negotiation process. Several negotiation protocols exist that support automated negotiation, including CNP [159], COPS-SLS [114], SNAP [43] and WS-Agreement [5].

The WS-Agreement specification provides a basis for defining services with Service Level Agreements (SLA). An SLA is an agreement between multiple participants that specifies terms of service (e.g. price, quality). WS-Agreement provides basic negotiation objects and a well-defined language for creating SLAs. This specification, however, provides only a basic protocol for SLA negotiation. This protocol supports only a single round of negotiation. For instance, a consumer requests a service; a provider accepts or rejects the request. If the provider rejects the request, the consumer cannot propose a new

request[1] or inquire the reason for rejection. This model of communication does not satisfy a bidirectional negotiation dialogue, as defined by this dissertation.

To resolve these issues, this dissertation presents the WS-Agreement Negotiation [174] specification. This specification extends the WS-Agreement specification with support for bidirectional, multiround negotiation with limited argumentation. WS-Agreement Negotiation defines two layers: *negotiation layer* and *agreement layer*. The negotiation layer supports bidirectional, multiround dialogue (e.g. offer, counter-offer). The agreement layer supports creation of SLA using existing WS-Agreement objects.

The WS-Agreement Negotiation specification meets many of the requirements of multiagent negotiation in open environments. However, it does not support symmetry of roles. This specification defines two main roles, server and client, with different abilities and permissions. For instance, the server role has greater access to negotiation data than the client role. The WS-Agreement and WS-Agreement Negotiation specifications are designed specifically for web services. Web services traditionally use client-server roles that are often asymmetric (i.e. a server has more data, access and abilities than a client). These services also traditionally follow a strict, asymmetric request-response model of interaction (e.g. a server is always reactive, never proactive).

Negotiation in open environments requires negotiation protocols that support flexible, symmetric roles. Such protocols support scenarios in which agents both consume and produce services. To address this issue, this dissertation presents several extensions to WS-Agreement Negotiation to enable negotiation in open environments.

For clarity, this dissertation will refer to the negotiation protocol specifications (NPS) in this chapter in the following way:

[NPS-0] The Web-Service Agreement (WS-Agreement or WSAG) specification is an existing specification developed by the Grid Resource Allocation Agreement Protocol (GRAAP) working group at the Open Grid Forum (OGF). This specification provides the basis for the following two specifications.

[NPS-1] The Web-Service Agreement Negotiation (WS-Agreement Negotiation or WSAN) specification presented in this dissertation was developed in collaboration with the GRAAP-WG at OGF. This specification addresses specific requirements for multiround negotiation. The research presented in this dissertation contributed to this specification to make

[1]The consumer cannot propose a new request in the same negotiation session. To create a new request, the consumer must create a new session and begin the negotiation process from the beginning.

the protocol both stateless and asynchronous. This specification is now an official OGF standard.

[NPS-2] This dissertation proposes several extensions to `NPS-1`. These extensions provide (1) a dual state machine, (2) session identifiers that enable data symmetry between participant roles and (3) explicit semantics to clarify intervals.

This chapter proceeds as follows. First, an overview introduces the main concepts of service negotiation. Secondly, this chapter provides an overview of negotiation protocols, including WS-Agreement (`NPS-0`). Thirdly, this chapter presents the WS-Agreement Negotiation protocol (`NPS-1`), including several extensions (`NPS-2`) for negotiation in open environments. Fourthly, this chapter describes an implementation of WS-Agreement Negotiation in the AgentScape middleware, including experimental results. Finally, this protocol is compared to related work.

3

3.1 Introduction

This section provides a detailed overview of service negotiation, starting with the core concepts of negotiation. This overview introduces: (1) various forms of negotiation, including auctions and bargaining; (2) the components of the negotiation process, including roles and cardinality, and (3) the concept of *utility*.

In addition, this section introduces Service Level Agreement (SLA). An SLA is a digital document that formalizes the negotiation process by specifying the terms of service between negotiating agents. SLAs are fundamental to the negotiation protocols discussed in this chapter.

Finally, this section introduces the challenges and requirements of automated negotiation. Requirements include a well-defined negotiation protocol and service specification language.

3.1.1 Service Negotiation

Negotiation is a bidirectional process by which one or more participants (e.g. a provider and a consumer agent), with possibly conflicting goals, together search for a mutually acceptable agreement [75]. In the simplest case, negotiation is a one-to-one interaction between two participants: a service provider and a service consumer. A single negotiation process, often referred to as a single negotiation session, is characterized by proposals, counter-proposals, trade-offs and concessions. Negotiation may focus on one or more issues, referred to as single or multiple issue negotiation [109]. Single issue negotiation,

for example, most often focuses on price. In contrast, multiple issue negotiation, focuses on more than one attribute, such as price, quantity and Quality of Service (QoS) attributes.

Negotiation takes several forms, including auctions and bargaining [107]. Generally speaking, an auction uses a centralized auctioneer to accept bids from all other negotiation participants, after which a winning bid is selected. Auctions may follow one of several bidding protocols, such as English, Dutch and Vickrey [14].

This dissertation focuses primarily on the second form of negotiation, known as bargaining. In general terms, bargaining consists of two or more participants exchanging offers. Each offer is evaluated using a set of personal preferences (e.g. utility functions) to determine the offer's value or score [137]. Various negotiation strategies are followed to increase the utility of an offer (e.g. maximize quality or minimize costs). Most strategies are rooted in the principles of game theory, which assumes that negotiation participants are self-interested and rational [14, 146].

A utility function is a method of adding up the relative weights of individual issues [167]. For instance, a specific consumer may evaluate offers from providers based on a combination of price, provider's reputation and location. For this particular consumer, price is the most important of these issues and location is the least important. Equation 3.1 presents a corresponding utility function. In this equation, the total utility (U_{total}) is calculated by summing the relative utilities of price (U_{price}), reputation ($U_{reputation}$) and location ($U_{location}$). Each issue is weighted according to the relative importance to the consumer.

$$\mathbf{U_{total}} = (U_{price} * 50\%) + (U_{reputation} * 40\%) + (U_{location} * 10\%) \tag{3.1}$$

Several general steps of negotiation are identified, including (1) offer specification, (2) offer submission, (3) offer analysis, (4) offer matching, (5) offer allocation and (6) offer acceptance [162]. In the first step, both negotiation participants (i.e. consumer and provider) specify negotiation intentions (e.g. range of services) and constraints (e.g. time limitations). In the second step, one participant (i.e. provider) sends an offer to the other participant (i.e. consumer). In the third step, the recipient analyzes the offer. This involves several checks to both syntax and semantics of the offer. Questions are posed, such as: *Is the document structure valid?, Does it contain all required data?, Are the requested services available?, Is the price reasonable?*

In the fourth step, offers are compared. For example, consumers compare available service offerings from a given provider to competing offers from

other providers. Each consumer attempts to find the "best" offer for the given situation. Once an offer is chosen, the provider of the requested service allocates the requested resources in the fifth step of the negotiation process. This prevents over-provisioning of services and reduces the likelihood of SLA violation. In the final step, the offer is officially accepted by both participants. This typically involves some formality to "sign" the document. In the digital domain, this is accomplished with nonrepudiable signing protocols, as discussed in Chapter 4. If successful, negotiation results in an agreement between the participants involved. This agreement takes the form of an SLA. The following section discusses SLAs in more detail.

The cardinality of negotiation specifies the number of participants involved in a particular negotiation session. A session may be one-to-one, one-to-many or many-to-many. An example of a one-to-one session is one consumer and one provider. Many-to-one or many-to-many sessions involve multiple providers or multiple consumers. For instance, a coalition of consumers may negotiate with a coalition of providers. Additionally, a single participant may be involved with multiple negotiation sessions at the same time. For instance, a consumer may negotiate with several providers in parallel, to compare offers.

Each participant assumes one or more roles during a negotiation session, such as provider and consumer. A provider provides a service and a consumer uses or consumes the service. In some environments, participants may assume more than one role at the same time. For instance, a typical scenario in Cloud computing is for a resource consumer to install private software and resell Cloud infrastructure as a separate service. An example of this is the Dropbox file hosting service[2]. Dropbox offers file hosting services to customers. In this relationship, Dropbox fills the role of provider. However, Dropbox has no private infrastructure, but rather combines hardware from Amazon Web Services[3] with proprietary software to fulfill customer needs. In this relationship, Dropbox fills the role of consumer.

Additional roles include intermediaries, including Match-Maker, Broker or Mediator [107]. These roles indirectly assist the negotiation process. The intermediary may assist by introducing consumers to suitable providers, based on matching consumer needs to provider offerings. The intermediary may also negotiate on behalf of one or more participants. For instance, a single intermediary represents a coalition of participants. Intermediaries may also assist if a conflict arises as discussed in Section 4.1.6.

Figure 3.1 illustrates several possible negotiation scenarios involving different cardinalities and roles. In the first scenario, a single consumer negotiates

[2] http://www.dropbox.com
[3] http://aws.amazon.com

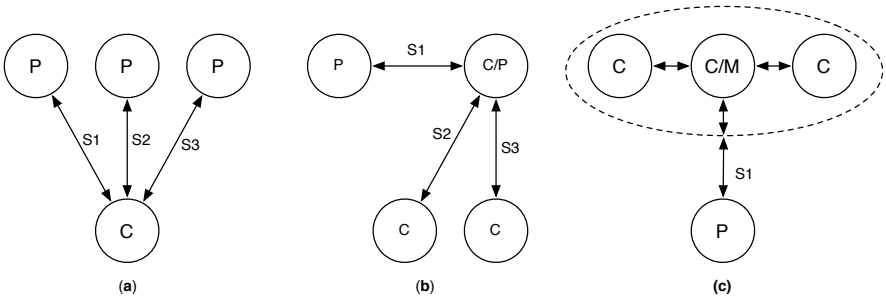

Figure 3.1: Cardinalities and roles of negotiation. (a) One consumer negotiates simultaneously with three providers. (b) A negotiation participant with both consumer and provider roles in separate negotiation sessions. (c) A consumer with the dual-role of mediator.

three separate sessions (S1, S2, S3) with three separate providers. In the second scenario, a provider offer services (S1) to a consumer. The consumer, in turn, resells these services to two separate consumers (S2, S3). In the final scenario, a coalition is formed by three consumers, in which a single consumer assumes the dual-role of mediator, representing the coalition. This single mediator negotiates a single session (S1) with a single provider.

3.1.2 Service Level Agreements

Service Level Agreements (SLA) are agreements between multiple participants that specify terms of service. They involve at least one provider and at least one consumer and specify the services that are provided. Traditionally, SLAs are written and signed between legal entities (e.g. between lawyers or other human actors), representing each of the participants involved. In recent years, attention has been given to automating this process [81, 94]. This dissertation focuses on SLA negotiation between software agents (see Section 2.3.1).

Automated SLA creation requires additional legally binding frameworks [20]. For the purpose of automation, several specifications exist to describe and negotiate SLAs, including the WSLA [96] and WS-Agreement [5] specifications. Section 3.2 describes WS-Agreement in more detail.

As an example, two agents negotiate an SLA for web-hosting. One agent provides access to a web-hosting service. Another agent is interested in this service to host a website. An SLA contains an explicit description of the service (e.g. name, URL). A period of validity may be negotiated in terms of time (e.g. hours, months) or in terms of activities (e.g. after completion of a specific task). In addition, the SLA contains the exact terms that comprise the service (e.g. 10 GB of disk space, 1 GB of network traffic).

An SLA document also includes Quality of Service (QoS) guarantees. QoS is expressed as a set of `(name, value)` pairs where `name` refers to a Service Level Objective (SLO) and `value` represents the requested level of service. An SLO specifies the particular characteristics of the service to measure, how to carry out measurements and actions to take after measurement. In the above mentioned example of a website hosting service, an SLA may contain the following pairs: `(uptime, greater than 99%)` and `(network response time, less than 2 seconds)`. An expanded list of SLA terms for online services is found in [2, 37].

An SLA also specifies what actions to take if an agreement is violated by one or more of the agents. Actions taken in response to violation may include cancellation of the service, monetary fines or demerits to an agent's reputation [141]. Section 4.1.7 discusses detection and penalization of violations in more detail.

3.1.3 Automated Negotiation

Automating the negotiation process reduces the amount of required input from (human) participants, enabling the negotiation process to become (semi) autonomous. Negotiation is often automated using software agents [11,13,21,25, 76,87,136,152,153,173]. To successfully negotiate in open environments, agents require a well-defined framework to support the negotiation process [11, 108]. A framework provides agents the basic tools and mechanisms required to discover and communicate with other agents. A framework also offers agents a well-defined *negotiation protocol* and *service specification language*. Furthermore, a framework offers agents a shared ontology that provides a formal definition of all terms of negotiation [52, 164].

Agents require a shared and clear understanding of how negotiation proceeds and which actions are possible during each step of the negotiation process. A negotiation protocol provides this understanding. A negotiation protocol explicitly states which actions are possible, the possible order of the actions, the roles of the various agents, the states of negotiation (e.g. nonbinding offer, binding agreement) and the acceptable transitions between states. A service specification language defines the structure of negotiation objects (e.g. offers, bids) such that all agents are able to fully describe and understand the subject of negotiation (e.g. specific services and attributes). Several formal standards exist that specify a particular negotiation protocol and service specification language, including FIPA and WS-Agreement [5]. The latter of these uses SLA as a core negotiation object. As this dissertation focuses on SLA as the central object of negotiation, the remainder of this dissertation

describes, extends and implements this specification. Section 3.2 describes WS-Agreement in more detail.

3.1.4 Conclusion

The previous sections provide an overview of the core components, processes and terminology of service negotiation. This includes the concept of SLA; a document specifying terms of service. SLAs are fundamental to negotiation processes.

Negotiation processes are automated with software agents. Automation requires well-defined structure, including a *negotiation protocol* and *service specification language*. The WS-Agreement specification provides a protocol and language using SLA as a core negotiation object. The following section discusses this specification in more detail. This dissertation extends this specification. Sections 3.3 and 3.4 discuss this extension in more detail.

3.2 Web Service Agreement

The Web Service Agreement specification (`NPS-0`) standardizes SLA creation in distributed environments [5]. The Grid Resource Allocation and Agreement Protocol (GRAAP)[4] Working Group of the Compute Area of the Open Grid Forum (OGF)[5] develops and maintains WS-Agreement. WS-Agreement defines (1) an SLA creation protocol, (2) the basic objects of negotiation and (3) a language to express these objects.

3.2.1 Protocol Specification

The WS-Agreement protocol (`NPS-0`) is based on a single round, message exchange. The exchange of messages consists of three steps. Figure 3.2 illustrates these steps. In the first step, a consumer requests from a provider an overview of available services. Upon receipt of request, the provider sends an Agreement Template object (as discussed in the following section) to the consumer. In the second step, the consumer analyzes the available services and chooses one or more options. The consumer then makes an offer to the provider to request this choice of services. An offer describes the chosen services and relevant attributes (e.g. price). In the final step, the provider analyzes the offer and decides either to accept or reject. If the provider accepts the offer, an SLA is created. If the provider rejects the offer, the protocol terminates.

[4] `https://forge.ogf.org/sf/projects/graap-wg`
[5] `http://www.gridforum.org/`

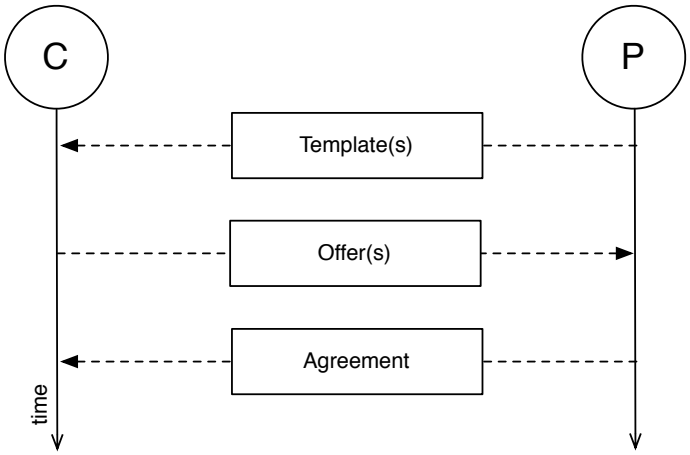

Figure 3.2: WS-Agreement SLA creation protocol. Adapted from [108].

Mobach extends the basic protocol with an additional acceptance/rejection phase [107]. In the original protocol, the responder (i.e. provider) accepts one offer from the initiator (i.e. consumer) and sends an Agreement to the initiator. The Agreement is finalized upon receipt. Mobach extends the protocol to include an additional acceptance/rejection phase allows the initiator to first consider the Agreement and then send an additional message to the responder to accept or reject the offer. The agreement phase is therefore a two-way exchange before an agreement is finalized.

3.2.2 Object Specification

WS-Agreement (`NPS-0`) specifies three basic objects of negotiation: *Agreement Templates*, *Agreement Offers* and *Agreements*. An Agreement Template provides an overview of available services. Providers use Agreement Templates to advertise available services and possible configurations for each service. For instance, an Internet Service Provider may advertise several packages (e.g. basic, medium, extreme) with different configurations of bandwidth, latency speeds and price. Optionally, a template has *Creation Constraints* that define the limits of offers based on the particular template. A constraint may be to limit the choice of services to a particular set or range. For example, a service may be limited to an enumeration of working days (e.g. Monday, Tuesday) or to a range of values (e.g. more than 1, less than 10).

A consumer uses an Agreement Offer to request a set of services proposed by a provider's Agreement Template. The consumer indicates the services and the attributes requested (e.g. preferred price, QoS). Continuing in the

example above, a consumer selects a particular internet package from the available choices.

An Agreement is created if both participants accept an Agreement Offer. Figure 3.3 depicts the basic structure of an Agreement (i.e. an SLA). The Context contains relevant information that defines a particular agreement, including the initiator (e.g. a consumer), responder (e.g. a provider), the time at which the agreement expires and the Template on which this particular agreement is based. Agreement Terms consist of *Service Description Terms* and optional *Guarantee Terms*.

A Service Description Term (SDT) defines a particular service with a name and description. Continuing in the example above, an SDT defines an Internet Connectivity package with 3 MB bandwidth for 50 euros per month. This information also includes information regarding where the service accessed. Guarantee Terms (GT) define optional information regarding the agreed quality objectives (QoS), the importance of a particular objective (Business Value) and the penalties to enforce if an objective is not met. Section 4.1.7 discusses penalties in more detail. In regard to the example above, a possible GT is network latency below 2 seconds and connectivity uptime above 99%.

3.2.3 Language Specification

The WS-Agreement specification (`NPS-0`) describes the above mentioned objects using the eXtensible Markup Language (XML)[6]. Figure 3.4 provides an XML version of the basic Agreement structure (introduced above). This example depicts the markup terms (e.g. wsag:Terms, wsag:Penalty) representing each component of a Template, Offer or Agreement. An XML document provides a machine readable rendering of each object. All automated analysis, decision making and negotiation strategies rely on understanding and expressing negotiation intentions through this medium. In addition to the markup terms defined in the official specification, WS-Agreement supports extensibility. Negotiation participants may add domain specific terms to objects, if necessary.

3.2.4 Single Round Negotiation

The single round nature of WS-Agreement (`NPS-0`) limits the dialogue between participants. The Agreement Initiator makes a proposal and the Agreement Responder responds with 'yes' or 'no'. If the proposal is rejected, no further explanation is given. If the Initiator wishes to continue, a new proposal is

[6] `http://www.w3.org/XML/`

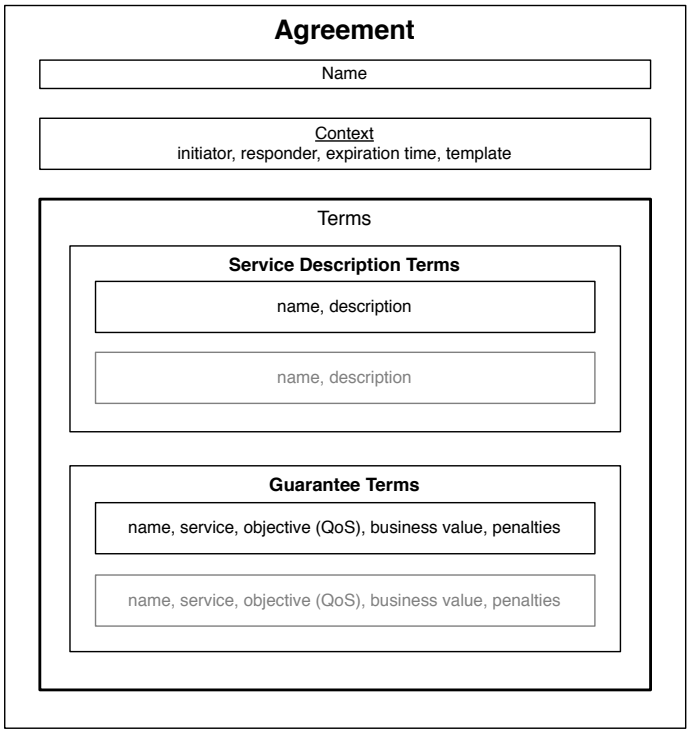

Figure 3.3: WS-Agreement SLA structure. Adapted from [5].

made. However, the Initiator has no knowledge as to the reasons for rejection, and thus, no guidance on creating a new proposal more likely of being accepted.

In multiround negotiation between participants, argumentation schemes convey reasoning or explanation for a given decision [7, 142, 155]. Structured argumentation allows participants (i.e. agents) to explicitly state the reasons for certain decisions (e.g. offer rejected because price is too high). Argumentation can also persuades a participant to accept a certain decision (e.g. a competitor is offering a better price) [169]. Formalized argumentation thus guides negotiation towards acceptable outcomes.

Combining formalized argumentation with a multiround negotiation protocol supports extensive negotiation discourse between participants. A series of offers and counter-offers between participants provides them the ability to iteratively reach an agreement. For example, during a particular negotiation session, a provider receives an offer. The provider agrees with all but one of the terms, namely, the term regarding price. The provider creates a counter-offer consisting of the original terms along with a new price. The consumer

```
<wsag:Agreement AgreementID="xs:string">
  <wsag:Name>xs:string</wsag:Name>
  <wsag:Context>
    <wsag:AgreementInitiator/>
    <wsag:AgreementResponder/>
    <wsag:ExpirationTime/>
    <wsag:TemplateName/>
  </wsag:Context>
  <wsag:Terms>
    <wsag:All>
      <wsag:ServiceDescriptionTerm>
        <wsag:Name/>
        <wsag:Description/>
      </wsag:ServiceDescriptionTerm>
      ...
      <wsag:GuaranteeTerm>
        <wsag:Name/>
        <wsag:ServiceScope/>
        <wsag:ServiceLevelObjective/>
        <wsag:BusinessValueList>
          <wsag:Penalty/>
          <wsag:Reward/>
        </wsag:BusinessValueList>
      </wsag:GuaranteeTerm>
      ...
    </wsag:All>
  </wsag:Terms>
</wsag:Agreement>
```

Figure 3.4: WS-Agreement XML language representation.

implies from this counter-offer that the provider is satisfied with all terms except price. If negotiation proceeds, the consumer focuses on this term (e.g. propose a slightly higher price than the initial proposal).

3.3 Web Service Agreement Negotiation

The WS-Agreement Negotiation specification (NPS-1) is the result of collaboration with and participation in the GRAAP working group of OGF. The research presented in this dissertation contributed to this specification to make the protocol both stateless and asynchronous. WS-Agreement Negotiation is a protocol for negotiating agreements between two participants [12, 174]. After passing rigorous review, this protocol is currently an official OGF standard[7].

[7]Version 1.0 available at: http://www.gridforum.org/Public_Comment_Docs/Documents/2011-03/WS-Agreement-Negotiation+v1.0.pdf

WS-Agreement Negotiation (NPS-1), presented in this dissertation, extends the existing WS-Agreement protocol (NPS-0)with an explicit negotiation protocol. The combination of these specifications has two layers of interaction: (1) *negotiation layer* and (2) *agreement layer*. Participants begin in the agreement layer. This layer consists of three phases: Template(s), Offer(s) and Agreement. These phases are depicted above in Figure 3.2.

If participants require multiround negotiation during the Offer phase, participants proceed to the negotiation layer. The negotiation layer supports multiround negotiation with limited argumentation. If negotiation results in an acceptable offer, participants return to the agreement layer to enter the Agreement phase.

WS-Agreement Negotiation defines (1) an SLA negotiation protocol, (2) additional objects of negotiation and (3) a language to express these objects. The language is based on and compatible with the WS-Agreement specification.

3.3.1 Protocol Specification

Essentially, the negotiation layer specified by the WS-Agreement Negotiation standard (NPS-1) defines the exchange of a series of offers during multiple rounds of negotiation. A series of offers and counter-offers regarding the negotiation of specific services between participants is referred to as a negotiation session. A negotiation session begins with one participant (e.g. consumer) making an offer to one or more participants (e.g. providers). A participant may respond to this offer by accepting, rejecting or proposing a counter-offer. The counter-offer is based on the previous offer, but may contain service terms deemed more acceptable (e.g. a lower price) than the previous offer. If an offer is accepted, the negotiation session terminates and the participants create an agreement using the agreement layer provided by WS-Agreement (NPS-0). An accepted negotiation offer is nonbinding. Binding of the agreement occurs elsewhere. If an offer is rejected, the session either terminates or continues with a new counter-offer with more acceptable terms.

Figure 3.5 illustrates negotiation. This figure portrays the interactions over time between a single consumer C and a single provider P. Faded lines indicate the existing agreement layer provided by WS-Agreement (NPS-0). The new negotiation layer adds an optional phase to the agreement layer and supports multiple rounds of sending, evaluating and responding to offers. Dotted lines indicated the negotiation session which encapsulates the series of offers and counter-offers between these two participants.

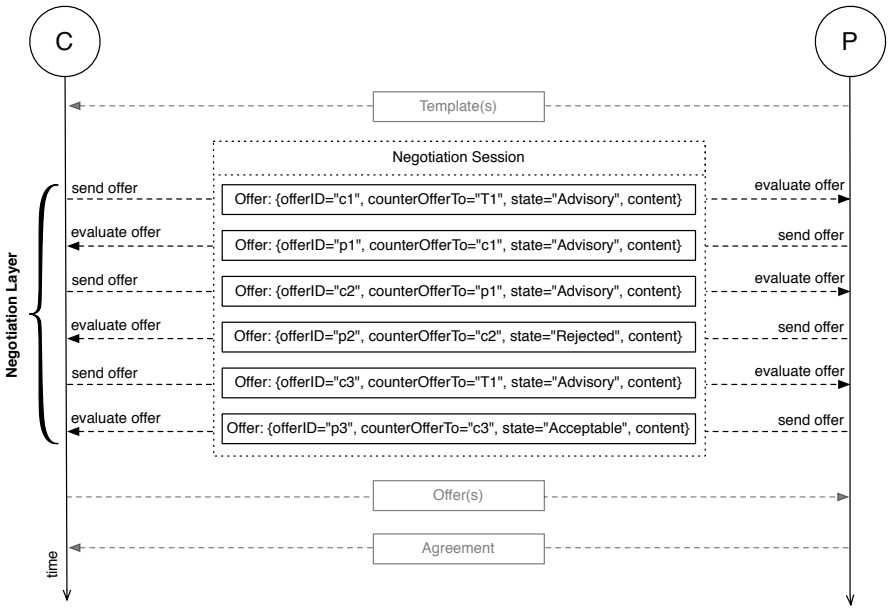

Figure 3.5: WS-Agreement Negotiation (`NPS-1`) multiround negotiation protocol. The existing agreement layer is depicted slightly faded. The new negotiation layer is indicated with the large curly bracket on the left side of the figure.

3.3.2 Object Specification

The WS-Agreement Negotiation specification (`NPS-1`) uses the same specification language as WS-Agreement (`NPS-0`), with several extensions to the offer document. An offer contains (1) an *offerID*, (2) a *counterOfferTo* field, (3) a negotiation *state* and (4) the offer content. The offerID uniquely identifies an offer within a given session (e.g. c1, c2, p1, p2). The counterOfferTo field contains the offerID of a previous offer in the session upon which the current offer is based. For example, if a consumer creates an offer *c2* in response to an offer from a provider *p1*, then this relationship is recorded in the counterOfferTo field. The negotiation state contains the current phase of negotiation. Finally, the offer content includes the negotiable service terms, such as SDTs and GTs.

As with WS-Agreement (`NPS-0`), the specification also allows for domain specific extensions. For instance, if a specific offer is rejected, the document may also contain a domain specific reason for rejection (e.g. price is too low).

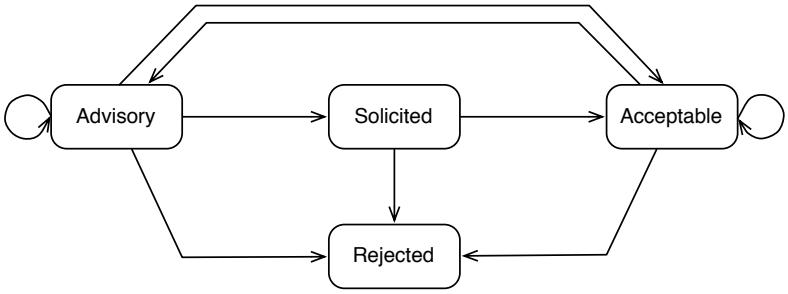

Figure 3.6: WS-Agreement Negotiation (`NPS-1`) state machine. Adapted from [174].

3.3.3 Negotiation State

Each offer contains a field indicating the current negotiation state. Figure 3.6 illustrates the possible state values and valid state transitions. The *Advisory* state indicates that the purpose of an offer is to gather information or test a participant's response. For instance, to explicitly elicit a provider's acceptable price range, an offer is sent containing a specified service, but no specified price. An offer in this state is not necessarily fully instantiated and may include empty terms or partially selected services.

If an offer is fully instantiated (e.g. no empty terms) and contains acceptable terms, a participant may place this offer in the *Acceptable* state. This state is nonbinding and only indicates that, if made, such an offer is likely accepted in the agreement layer.

If a participant chooses to reject an offer, that participant returns the offer after changing the state to *Rejected*. This indicates that future offers should not refer (e.g. the counterOfferTo field) to this offer. For a specific offer, the rejection state is terminal. However, negotiation may continue by creating a new offer, based on some other previous offer.

If a participant wishes to converge a negotiation session, an offer is sent in the *Solicited* state. This state essentially requests that the responding participant provide a 'yes' or 'no' answer. Thus, counter-offers must be fully instantiated and in either the acceptable or rejected state.

3.3.4 Session Rollback

During the course of a multiround negotiation session, many offers are exchanged. Each offer includes a record of the relationship with a previous offer (e.g. counterOfferTo). This record of the relationships between offers serves to

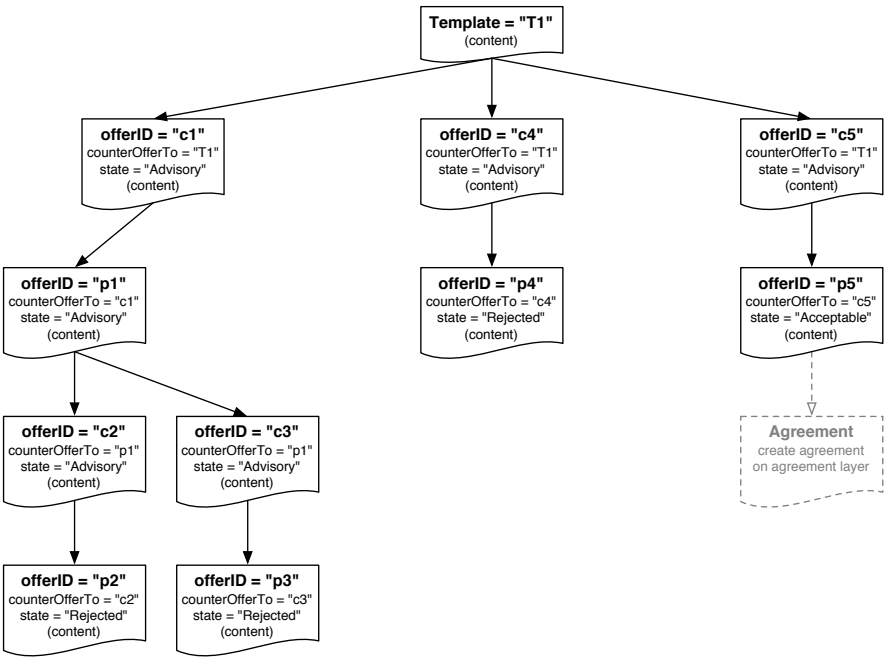

Figure 3.7: Negotiation offers arranged in tree structure. Adapted from [174].

organize offers within a negotiation session. The record of offers and counter-offers may be analyzed to learn from past interactions and improve negotiation strategies.

One possible organization of offers is a tree structure. In terms of a tree, the *root* is the Template upon which all subsequent offers are based. When an offer is made based on a previous offer, the offer is referred to as a *child* and the previous offer is referred to as its *parent*. A particular sequence of related offers is grouped together as a *branch*. Offers to which no counter-offers are made are referred to as *leaves*. Figure 3.7 illustrates this structure.

This figure provides an example interaction between a single consumer and a single provider. Each offer contains a unique offerID that indicates the author and order of that particular offer. For example, offer *c1* is the first offer created by the consumer and *p4* is the fourth offer created by the provider.

If a particular offer is rejected, participants may choose to immediately terminate the negotiation session. Alternatively, participants may choose to rollback to a previous negotiation round and continue negotiation with a different offer. For example, a consumer makes offer *c2* and a provider rejects this offer. By doing so, the provider has indicated that this particular branch of negotiation is unacceptable. The consumer responds by creating a new

branch. This occurs when the consumer performs a rollback to a previous offer *p1* and creates a new offer *c3* with content that differs from the rejected offer.

A rollback occurs again when offer *c3* is rejected. The consumer then performs a rollback to the original template *T1* and creates a new offer *c4*. When this offer is rejected, another rollback leads to the creation of *c5*. This offer's terms are acceptable to the provider and the negotiation session ends so an agreement may be created in the agreement layer.

The ability to rollback allows for negotiation to continue despite the rejection of a particular offer or branch. Unique offerID and counterOfferTo fields allows all participants to organize offers within a session. This organization, in turn, makes it possible to identify (and ignore) previously rejected offers or negotiation branches.

3.3.5 Dual State Machine Extension

Figure 3.6 depicts the current state machine of WS-Agreement Negotiation (`NPS-1`). This single state machine models both the current negotiation offer and, by proxy, the entire negotiation session. As such, this state machine has several drawbacks. One drawback is that this state machine has no terminal state. As such, negotiation is not terminated transparently and officially. Rather, negotiation ends when one participant chooses to ignore new offers from another participant.

This occurs because the most logical terminal state *Rejected* is only a terminal state for the particular offer that has been rejected. The session does not end, as the rejected participant is always permitted to perform a rollback and create a new offer with different terms. The *Solicited* state forces a negotiation to converge to a 'yes' or 'no' decision. If a negotiation session is arranged in a tree structure, as in Figure 3.7, then the current state machine only models a single negotiation branch, not the entire negotiation tree.

One solution is to maintain dual state machines. Figure 3.8(a) shows the current WS-Agreement Negotiation (`NPS-1`) state machine. This first machine models the "local" state of the current offer and negotiation branch. Figure 3.8(b) shows a proposed second state machine (`NPS-2`). This second machine models the "global" state of the session or entire negotiation tree.

The session state machine contains four valid states: *Negotiating*, *Request*, *Demand* and *Finalized*. As the negotiation session may always rollback and explore different negotiation branches, it essentially remains in a single state. This state is referred to as the Negotiating state. If a negotiation reaches a stage that would likely result in an accepted agreement (e.g. an acceptable

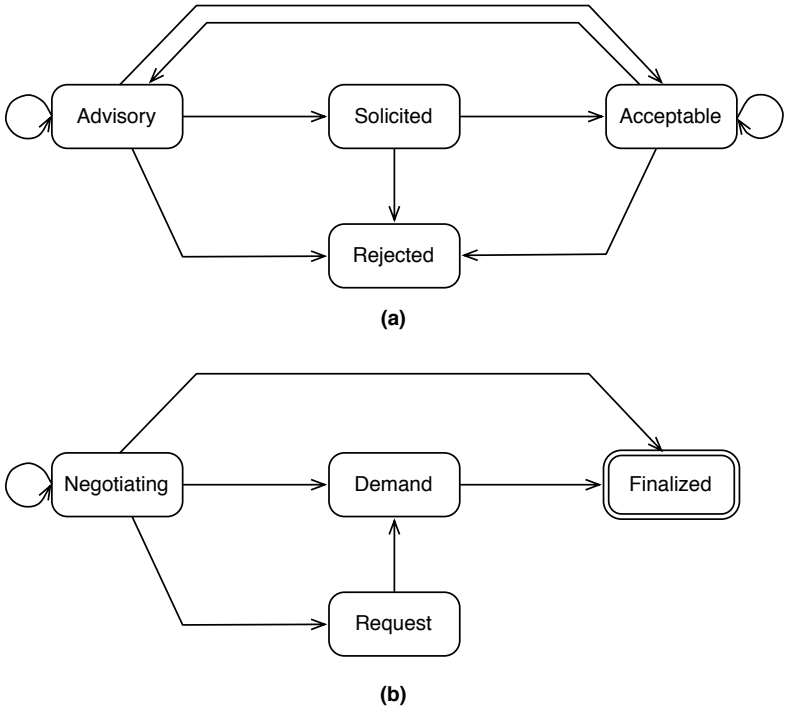

Figure 3.8: Dual state machines. (a) Original, offer state machine (from `NPS-1`). (b) Extended, session state machine (from `NPS-2`).

offer), then the session state should reflect this by transitioning to the Finalized state.

In the event that a participant wishes to force the session to converge to a 'yes/no' scenario without the option of rollback, the session transitions to the Request or Demand state. If a participant sends an offer in the Request session state, the responding participant must reply with an offer that can immediately be accepted or rejected without further negotiation (e.g. fully instantiated). If a participant sends an offer in the Demand session state, the responding participant must reply with an offer in the Request state. The essentially requests to see the final offer before accepting or rejecting it. The Request and Demand states allow for either the negotiation initiator or responder to converge the negotiation tree to termination.

In summary, the Negotiating state is reflexive. The Finalized state is terminal. The Request and Demand states force a negotiation session to convergence to an explicit and final termination.

Offers contain both states: *session state* and *offer state*. Table 3.1 shows the valid dual state combinations. Session (tree) state is listed along the

Table 3.1: Valid dual state combinations.

	Advisory	*Solicited*	*Acceptable*	*Rejected*
Negotiating	✓	✓	✓	✓
Request	✓	✓	✓	✓
Demand			✓	
Finalized	✓	✓	✓	✓

vertical axis and offer (branch) state is listed along the horizontal axis. A few of the state combinations have special significance:

Finalized & Acceptable - This dual state combination signifies the end of the negotiation session with a high chance of an acceptable agreement in the agreement layer. The negotiation session is successful.

Finalized & Advisory - This combination signifies the end of the negotiation session with a low chance of an acceptable agreement. The offer may not be acceptable or even fully instantiated (e.g. empty terms remain). The negotiation session is unsuccessful.

Finalized & Rejected - This combination signifies the end of the negotiation session with zero chance of an acceptable agreement. The negotiation session is unsuccessful.

Negotiating & Rejected - The current negotiation branch is terminated, but the negotiation session may continue with a rollback and new offer.

3.4 Agent Negotiation in Open Environments

The new WS-Agreement Negotiation specification (`NPS-1`) enables multiround negotiation with limited argumentation. However, when agents represent participants during negotiation in open environments, several specific challenges arise that are not addressed in this specification. This dissertation proposes several extensions (`NPS-2`) to the new protocol to address these challenges, including symmetry and dynamism of multiagent negotiation.

As discussed in Section 1.1, open environments have additional considerations regarding the availability and security of a negotiation framework. A negotiation framework must be robust against malicious attacks from dishonest negotiation agents or other, external agents. These considerations are

addressed, in part, with service monitoring, auditing, cryptographic primitives and decentralization of negotiation components. Chapter 4 discusses these topics in more detail.

Agent communication is modeled around asynchronous communication between autonomous systems, such as peer-to-peer (P2P) relationships [122]. Agents communicate via messages passing that is less strict than the request-response paradigm of request-response (i.e. client-server) interaction. For example, an agent may send more than one message to another agent before receiving a response[8]. Multiagent negotiation requires flexible negotiation protocols that support multiple interaction models, such as unsolicited offers (i.e. response before request).

An additional challenge of negotiation in open environments is the dynamic nature of roles. In contrast to well-defined, static roles of traditional web services (e.g. the client-server relationship), agents may change roles dynamically to adapt to a given situation (e.g. environmental or policy changes). For instance, in energy markets, distributed generation of energy (e.g. a solar panel on a roof) allows a consumer to sell overcapacity. Thus, a consumer agent can also become a producer agent.

In these environments, negotiation protocols must allow dynamic roles without requiring fundamental changes. Changing roles, therefore, should not first require fundamental changes in permissions, additional data, methods, libraries, etcetera. For instance, if all negotiation data is stored by a provider, this data would need to first be transferred to a consumer before that consumer may assume the role of provider. If a provider has additional permissions or actions, time is needed to load additional libraries for a consumer to become a provider. In open environments, changes in roles must occur seamlessly to prevent disruption of the negotiation process.

One approach to supporting these changes is to design protocols with *symmetry*. A symmetric protocol grants all participants equal actions, permissions and (data) access, regardless of negotiation role (e.g. consumer, provider). Actions, such as negotiation initiation and offer creation, are the same for consumers and providers. Permissions and access to data are the same, regardless of role. Negotiation data, including the history of messages (e.g. offers) and current state should be symmetric between negotiating participants. Rather than a single participant (e.g. the provider role) maintaining all data, data should be maintained equally by all participants during a negotiation session. Symmetric negotiation protocols allow participants to change roles without

[8]This is not possible following a request-response protocol as one agent must first wait for a reply before sending a second request.

requiring fundamental changes (e.g. redistribution of data, new permissions, new objects or methods).

Storing all data with a single participant, provides that participant with a higher level of access and thus more power/control over that data. For instance, a provider may change locally stored negotiation data to reach a favorable outcome (e.g. delete or modify certain offers). Distributing the negotiation data across all participants removes this imbalance of power. Section 3.4.1 discusses an approach to distributing negotiation data symmetrically between negotiating participants using a *Session Identifier*.

Another challenge of open environments concerns shared ontologies, such as those of the Agent Communication Language (ACL) specification from FIPA [56]. Ontologies define the vocabulary of negotiation terms that are shared between negotiation agents [50,77,166]. For instance, an ontology may define precisely how a certain metric (e.g. bandwidth) is measured or what is precisely meant by a `'CPU'` Or `'RAM'`. Ontologies may also provide additional clarification to negotiation issues otherwise left underspecified. Specifically, service terms that comprise an interval of values (e.g. between 50 and 500) are often underspecified during automated negotiation, as discussed in Section 3.4.2.

This dissertation assumes that shared ontologies are inherently static and unable to dynamically adapt to changes over time (e.g. new prices, new products). Therefore, shared ontologies must be fully defined before negotiation begins. Furthermore, ontologies are commonly limited to a specific domain, such as energy [49], e-commerce [164] or crisis management [131]. As such, these are not well-suited to dynamic, open environments in which negotiable objects and relationships may change often. Automated negotiation in such environments requires a different approach.

One possible approach is ontology matching [157]. This process attempts to find corresponding terms between two or more, independent ontologies. For instance, a provider has one vocabulary containing the term `'Internet Provider'` and a consumer has a separate vocabulary containing the term `'Internet Service Provider'`. It is the task of ontology matching to link these two syntactically different but semantically identical terms. Ontology matching is a nontrivial challenge [156]. An alternative approach is to add additional *Interval Semantics* to the negotiation protocol [28,97]. Section 3.4.2 discusses this approach in more detail.

3.4.1 Session Identifier

The new WS-Agreement Negotiation specification (`NPS-1`) defines a *negotiation instance* object [174]. A negotiation instance is maintained by a provider

and contains all information relevant to a given negotiation session, such as offers and state information. Each negotiation instance is identified by a unique *Endpoint Reference* (EPR) as defined by the WS-Addressing specification [64]. An EPR contains a Uniform Resource Identifier (URI) that explains how a resource is accessed (e.g. a URL accessed via HTTP). EPRs also identify specific services and agreements (e.g. SLA documents). This approach to identifying services is appropriate for the area of web services based on the static, asymmetric roles of server and client. However, services for Multi Agent Systems (MAS) require a different approach.

In the MAS approach, agents are equals (e.g. peers). Roles are dynamic. Rather than a single role (e.g. a provider) maintaining all session data, the negotiation instance is distributed across negotiation agents. This is similar to the concept of replicated objects [165]. Each agent has a local copy of the negotiation instance. Unique identifiers link two related, but separate instances stored at different locations and controlled by different agents [165]. Instances are updated based on the information (e.g. state) stored in received offers.

Each participant, represented by an agent, thus maintains a separate, but equivalent, negotiation instance. Both consumers and providers have equal access to the negotiation instance containing the history of offers and negotiation state. As there is no single negotiation instance, an EPR is not necessary. Instead of an EPR, this dissertation proposes (`NPS-2`) that each negotiation message is labeled with an additional session identifier: a *Session ID*. This identifier enables a receiving agent to associate a particular negotiation offer to the correct negotiation instance.

Session IDs organize multiple, simultaneous negotiation sessions. Agents exchange asynchronous messages. Messages are stored in message buffers until the agent reads them and responds. Messages may arrive in the buffer out of order. Messages from separate, concurrent negotiation sessions may arrive simultaneously. This is in contrast to synchronous function calls associated with traditional web services. Using the included Session ID, each message (e.g. negotiation offer) is correctly organized by an agent itself. Figure 3.9(a) illustrates the role of session identifiers in each negotiation message from three separate negotiation sessions. This identifier is unique and is known by all agents to a given negotiation session. The Session ID is agreed upon before negotiation begins.

In some cases, multiple sessions are logically related. For instance, an agent simultaneously negotiates the sale of one car and the purchase of another. Regarding the sale of the first car, the agent negotiates with two potential buyers.

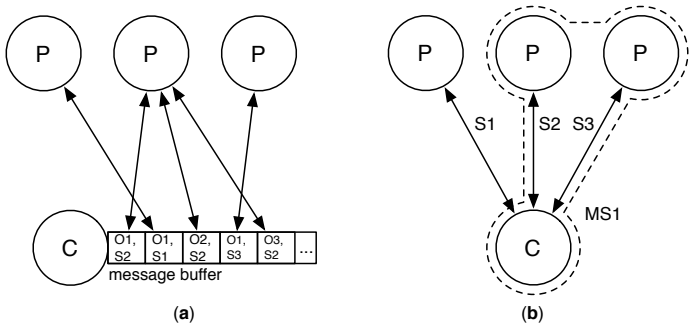

Figure 3.9: An illustration of the application of session identifiers (from NPS-2): (a) Three separate negotiation sessions with unique identifiers. (b) Logical grouping of separate sessions.

The agent logically groups these separate, but related, negotiation sessions using a *Multisession ID*. The Multisession ID is unique to the agent and is not necessarily known to the two potential buyers. Figure 3.9(b) illustrates the application of a Multisession ID: MS1.

3.4.2 Interval Semantics

During automated negotiation of services, autonomous agents use utility functions to evaluate the terms of negotiation, as discussed in Section 3.1.1. These terms include discrete values, such as *{Nuclear, Coal, Gas}* for energy suppliers or intervals of values, such as *{between 50 and 500}*. Evaluating utility of a discrete value is well understood [137]; however, evaluating utility of an interval of values is an area of ongoing research [167]. If unspecified, agents can possibly interpret intervals of values incorrectly. If unspecified (or underspecified), it may be unclear: (1) if the choices are exclusive or inclusive; (2) if an interval embodies a continuum of real numbers or a subset of natural numbers; (3) if one value may be chosen or multiple, or (4) if a sub-interval may be chosen or multiple sub-intervals. Automated negotiation requires clear semantics to correctly interpret intervals and compute utility. Neither the WS-Agreement (NPS-0) nor WS-Agreement Negotiation (NPS-1) specifications contain semantics to clarify intervals; therefore, this dissertation proposes additional semantics (NPS-2).

Figure 3.10 demonstrates the ambiguity of intervals with an example service offer. An offer lacking explicit interval semantics requires multiple assumptions. The assumptions made in this example include that *Provider* is an exclusive choice, as a contract in this scenario is either signed with one provider or the other, but not both. Another assumption is that the *Source* is

```
                    TEMPLATE

Base Rate = {0 - 100}
Quantity = {0 - 10000}
Provider = {A, B, C}
Sources = {Nuclear, Coal, Gas, Wind, Solar}
Green Percent = {0 - 100}
Availability = {75 - 100}
CO2 Compensation = {green investment}
Buy-back Rate Factor = {50 - 500}
```

Figure 3.10: Example resource offering with intervals. Adapted from [28].

not exclusive, as a contract may contain both solar energy during the day and coal energy during the night. More underspecification becomes apparent with the intervals *Base Rate* and *Quantity*. As Base Rate represents a monetary price, the assumption is that this interval is continuous with a precision of two or more digits. In contrast, the quantity of kilowatt hours is not typically specified with such a level of precision and this interval may actually only contain discrete choices in increments of 1000. These semantics, however, are not explicit and could cause incorrect assumptions, leading to a suboptimal or unacceptable negotiation offer.

Exclusive choices may describe some intervals. For instance, *Base Rate* starts at zero, yet this is not a valid choice, but rather an exclusive lower limit. The first valid choice may actually be 1 or 0.5 or some other positive number. When multiple choices are presented, the order of the choices may have meaning. For instance, *Source* may be ordered according to price, carbon emission, or preference. In contrast, when there is no order, it may be useful to express this fact explicitly, as well.

Often relationships and dependencies between choices require specification. For instance, some options may be inclusive, such as *Solar* energy may only be chosen in combination with a second energy source. Similarly, relationships between terms are important. For instance, if *Nuclear* energy is chosen, then only providers *A* and *B* are available. For intervals, the higher the *Availability*, the higher the *Price*. While these relationships could conceivably be derived from several rounds of negotiation, making them explicit could make for faster negotiation.

3.4.2.1 Expressing Intervals

Table 3.2 summarizes several issues often underspecified in automated negotiation. Each issue requires clear notation to convey the correct meaning. This

Table 3.2: Underspecified issues in automated negotiation.

Ordered or Unordered	Are multiple values ordered or unordered? If ordered, what is the meaning of the order?
Inclusive or Exclusive	Are the limiting values of an interval inclusive or exclusive?
Continuous or Discrete	Is an interval continuous or discrete? If continuous, to what precision? If discrete, what are the increments?
Value or Interval	Should choices be in the form of a single value or a sub-interval? How many of each?
Preference	Is there a preference for one choice above another?
Indifference	Is an agent indifferent to the value of a certain term?
Relationships of choices	Are there relationships between multiple choices?
Relationships of terms	Are there relationships between different terms?

notation is added to service offers and subsequent responses to indicate the exact meaning of a term to facilitate correct interpretation and evaluation.

Figure 3.11 shows the same resource template as before, but with added semantics. To differentiate an ordered list from an unordered list, an ordered list is surrounded by '$<$' and '$>$', whereas an unordered list is surrounded by '{' and '}'. Standard mathematical notation indicates whether an interval's limits are inclusive or exclusive. This requires a '(' or ')' for inclusive and a '[' or ']' for exclusive. The symbol '$***$' indicates indifference. All other issues use annotations that take predefined values.

Whether an interval is continuous or discrete is indicated with the annotation '`CD`' that takes a letter and number as its value. If continuous, the letter '`C`' is followed by a number indicating the precision. If discrete, the letter '`D`' is followed by a number indicating the size of the increments.

Whether an agent should choose a value or interval is indicated with the annotation '`VI`' that takes a letter and number as its value. If a value, the letter '`V`' is followed by the number of values that may be chosen. If an interval, the letter '`I`' is followed by the number of sub-intervals that may be chosen.

The '`PC`' annotation indicates preference between choices. The order of values conveys order of preference.

```
                                TEMPLATE

Base Rate = (0 - 100] | CD:C5, VI:I1
Quantity = (0 - 10000] | CD:D100, VI:V1
Provider = {A, B, C} | VI:V1
Sources = {Nuclear, Coal, Gas, Wind, Solar} | VI:V2, RC:Wi:OR:So
Green Percent = [0 - 100] | CD:C0, VI:I1, RT:DECREASES:Availability
Availability = [75 - 100) | CD:C0, VI=V1, RT:DECREASES:Green Percent
CO2 Compensation = {green investment} | VI:V1, RT:ONLY:A
Buy-back Rate Factor = [0.1 - 4] | CD:C1, VI:I1

                                OFFER

                Base Rate = [5.5 - 12]
                Quantity = {5000}
                Provider = {A}
                Sources = <Solar, Gas> | PC:YES
                Green Percent = [***]
                Availability = {99}
                CO2 Compensation = {green investment}
                Buy-back Rate Factor = [1 - 2]
```

Figure 3.11: Resource offer and response with added semantics (NPS-2).

The 'RC' annotation indicates that a relationship exists between two choices. This takes the value of 'TERM:RELATIONSHIP:TERM' where 'RELATIONSHIP' is a predefined term, such as 'INCREASES' or 'REQUIRES'. Similarly, the 'RT' annotation indicates relationships between two terms. This takes the value of 'RELATIONSHIP:TERM' and uses a set of predefined relationships, such as 'AND' or 'ONLY'.

Figure 3.11 is interpreted as follows: *Base Rate* is an interval that excludes the lower limit and includes the upper limit. Furthermore, it is continuous to five digits past the decimal point and one sub-interval may be chosen. *Quantity* is also an interval that excludes the lower limit and includes the upper limit. Furthermore, it is discrete with increments of 100 and a single value may be chosen. *Provider* is an unordered list and only one value may be chosen. *Sources* is an unordered list and two values may be chosen. Furthermore, either *Wind* or *Solar* may be chosen, but not both. *Green Percent* is an interval of continuous natural numbers with inclusive limits. A single sub-interval may be chosen and as this value increases, *Availability* decreases. *Availability* is an interval with an inclusive lower limit and an exclusive upper limit. Furthermore, it is continuous with zero digits of precision and a single value may be chosen. *CO2 Compensation* is only available from provider "A". Finally, *Buy-back Factor* is an interval with inclusive upper and lower

limits. Furthermore, it is continuous with one digit of precision and a single sub-interval may be chosen.

The offer made based on the template also uses added semantics. *Base Rate* contains an interval with inclusive limits. *Sources* is an ordered list ordered by preference. Furthermore, the offer indicates indifference to the value of *Green Percent*.

3.4.2.2 Expressing Intervals in WS-Agreement

Both WS-Agreement (`NPS-0`) and WS-Agreement Negotiation (`NPS-1`) specifications express negotiation objects, such as Templates and Offers, using XML, as discussed in Section 3.2. The interval semantics this dissertation proposes (`NPS-2`) are also expressed in XML. Continuing the earlier example of energy provision, semantic annotations take the form of XML tags and are added to the XML schema to resolve underspecified issues. Figure 3.12 and Figure 3.13 show the same service template and offer, respectively, using XML structure based on WS-Agreement (`NPS-0`). Two agents negotiate the provision of energy. The provider advertises the available choices in a template using additional semantic tags, as introduced above.

The *min-* and *maxExclusive* tags replace the '(' and ')' symbols. The *min-* and *maxInclusive* tags replace the '[' and ']' symbols. The *baseRate* and *quantity* items illustrate these tags. An additional *ordering* tag replaces '{' and '<' to express ordering. Additionally, each element in the list uses a "rank" value with ascending order as XML does not natively support ordering of elements. The *sources* item of the offer illustrates this value. When an agent wishes to indicate preferential ordering, the agent modifies the value of this tag in the offer.

3.4.2.3 Use case scenario

A use case scenario demonstrates the applicability of interval semantics in a different domain. A single consumer C and a single provider P negotiate the sale of an automobile. Figure 3.14 provides an overview of the negotiation process. The consumer first requests a template *T1* that advertises all possible choices, including Make and Price. This template includes additional notation that describes the semantics of each interval of options. This notation states that only one value may be chosen for Object, Color and Make. Furthermore, a single interval may be chosen for Price, starting at 1.1k. The template lists all options, but describes exclusive options (e.g. diesel or petrol) using the notation. Finally, the template explicitly states the relationship between Power and Taxes (e.g. road tax, emissions tax).

```
                    TEMPLATE

<wsag:Item wsag:name="baseRate" CD="C5" VI="I1">
  <minExclusive="0"/>
  <maxInclusive="100"/>
</wsag:Item>
<wsag:Item wsag:name="quantity" CD="D100" VI="V1">
  <minExclusive="0"/>
  <maxInclusive="10000"/>
</wsag:Item>
<wsag:Item wsag:name="provider" VI="V1">
  <list ordering="NONE">
    <enum value="A"/>
    <enum value="B"/>
    <enum value="C"/>
  </list>
</wsag:Item>
<wsag:Item wsag:name="sources" VI="V2">
  <list ordering="NONE">
    <enum value="Nuclear"/>
    <enum value="Coal"/>
    <enum value="Gas"/>
    <enum value="Wind"/>
    <enum value="Solar"/>
  </list>
  <RC="Wi:OR:So">
</wsag:Item>
<wsag:Item wsag:name="greenPercent" CD="C0" VI="I1">
  <minInclusive="0"/>
  <maxInclusive="100"/>
  <RT="DECREASES:availability"/>
</wsag:Item>
<wsag:Item wsag:name="availability" CD="C0" VI="V1">
  <minInclusive="75"/>
  <maxExclusive="100"/>
  <RT="DECREASES:greenPercent"/>
</wsag:Item>
<wsag:Item wsag:name="co2Comp" VI="V1">
  <enum value="green-investment"/>
  <RT="ONLY:A"/>
</wsag:Item>
<wsag:Item wsag:name="buyBackFac" CD="C1" VI="I1">
  <minInclusive="0.1"/>
  <maxInclusive="4"/>
</wsag:Item>
```

Figure 3.12: WS-Agreement template with interval semantics (from NPS-2).

```
                OFFER

<wsag:Item wsag:name="baseRate">
  <minInclusive="5.5"/>
  <maxInclusive="12"/>
</wsag:Item>
<wsag:Item wsag:name="quantity">
  <enum value="5000"/>
</wsag:Item>
<wsag:Item wsag:name="provider">
  <enum="A"/>
</wsag:Item>
<wsag:Item name="sources" PC="YES">
  <list ordering="PREFERENCE">
    <enum value="Solar" rank="0"/>
    <enum value="Gas" rank="1"/>
  </list>
</wsag:Item>
<wsag:Item name="greenPercent">
  <enum value="***"/>
</wsag:Item>
<wsag:Item name="availability">
  <enum="99"/>
</wsag:Item>
<wsag:Item wsag:name="co2comp">
  <enum="green-investment"/>
</wsag:Item>
<wsag:Item wsag:name="buyBackFac">
  <minInclusive="1"/>
  <maxInclusive="2"/>
</wsag:Item>
```

Figure 3.13: WS-Agreement offer with interval semantics (from NPS-2).

After receiving the template, the consumer creates the first offer *C1*. As the consumer is unconcerned with the Color, Options or Taxes, this indifference is reflected by the '$***$' notation. Instead of choosing only one Make, the consumer chooses three, but indicates that they are ordered according to personal preference. This knowledge guides the provider to a more attractive offer sooner and, thus, a quicker sale.

Once the provider receives *C1*, it creates a counter-offer *P1*. All choices are now limited to the car Object. The consumer indicates that Make is preferably 'Audi', but there are no Audis available for the given price range, so the provider removes this from the counter-offer. This limits the Color to blue and only the available Options, Power and Taxes are included in the counter-offer.

The negotiation process continues with offers *C2* and *P2*. Each offer becomes more instantiated and leaves fewer choices with each iteration. Using

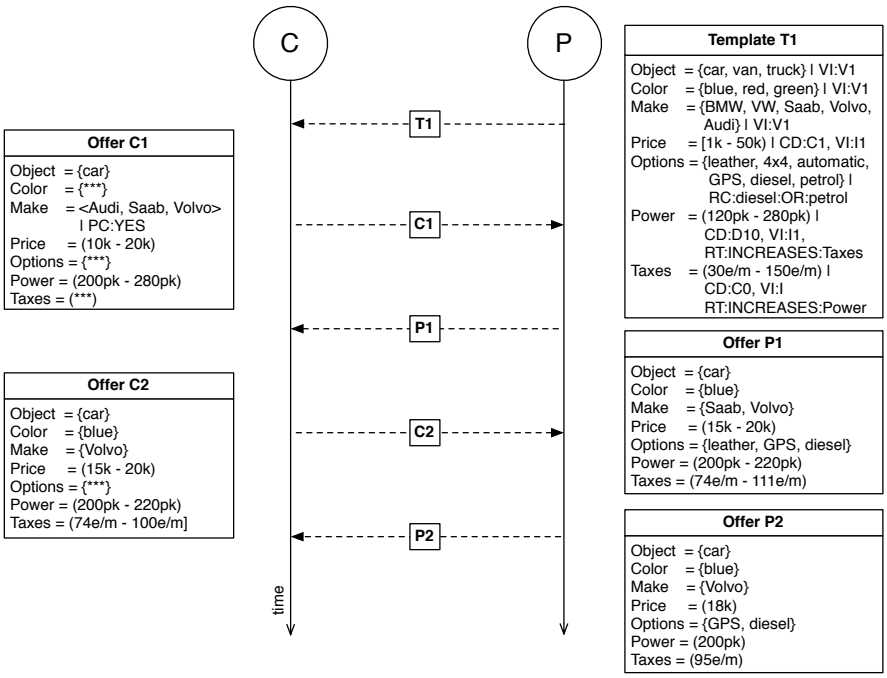

Figure 3.14: Interval semantic enhanced negotiation process.

notation to convey the semantics, the consumer searches through the possibilities to discover the most preferred options (e.g. offer *P2*).

3.5 Negotiation Protocol Implementation

The new WS-Agreement Negotiation (`NPS-1`) specification is implemented in the AgentScape [121] middleware. AgentScape is a distributed, Multi Agent System (MAS) for development and deployment of distributed applications. The framework offers a structured interface to shared services (e.g. directory services). Section 2.3.2 discusses AgentScape in more detail.

Agents represent providers and consumers during service negotiation using the WS-Agreement Negotiation protocol. The WSAN Service manages access to the protocol. Agents register with this service to access a standardized interface to relevant negotiation objects[9] and methods. Section 3.5.1 discusses selected objects and methods.

[9]The core objects (e.g. templates, offers and agreements) of this implementation are slightly modified versions from the WS-Agreement Negotiation for Java (WSAG4J) API written by Oliver Wäldrich of the Fraunhofer Institute for Algorithms and Scientific Computing. Source code available at: `http://wsag4j.sourceforge.net/`

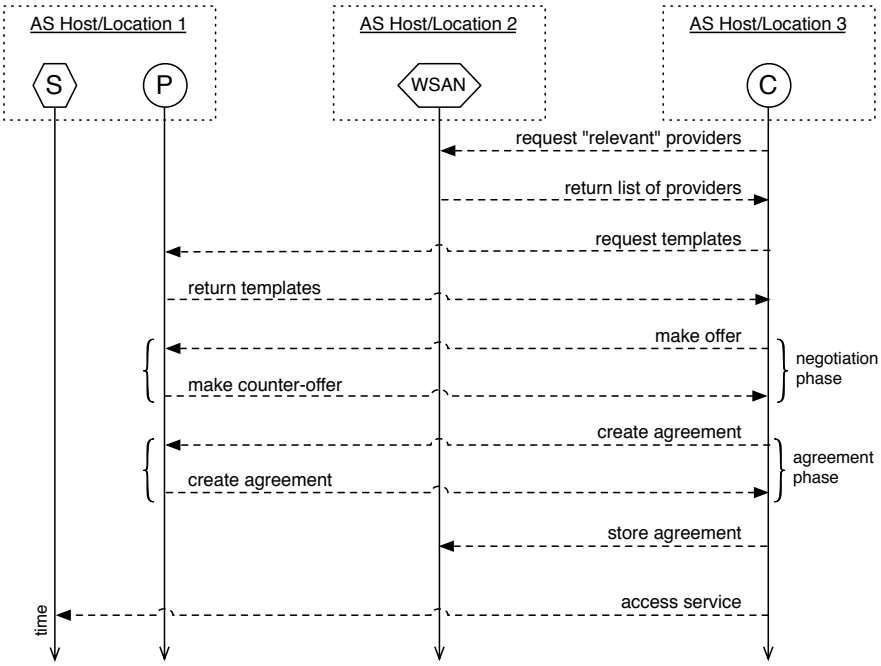

Figure 3.15: WS-Agreement Negotiation (`NPS-1`) protocol deployment in AgentScape.

Consumers, providers and the WSAN Service can be deployed in a single Location or distributed across separate Hosts or Locations. Figure 3.15 illustrates a negotiation process using the WSAN Service. A negotiation process begins when a consumer requests a list of providers from the WSAN Service. The consumer provides selected keywords to limit the list to only relevant providers (e.g. energy, electricity, power). Once a list of relevant providers is received, the consumer selects one or more providers and contacts them directly using the negotiation methods provided by the WSAN Service. The consumer requests and receives a list of Templates based on the selected keywords. Then the consumer chooses one or more Templates and enters into the negotiation phase. In the negotiation phase, the consumer uses the negotiation methods to create, send and evaluate one or more (counter) offers until an acceptable offer is reached. Once such an offer is reached, the consumer enters the agreement phase. This phase consists of a three-way handshake to confirm an Agreement. The Agreement is then stored at the WSAN Service.

Note that either agent (i.e. consumer or provider) may initiate the agreement phase. For instance, if a provider reaches an acceptable offer in the negotiation phase and wishes to end further negotiations, then it initiates the

agreement creation process. In addition, either agent may initiate the negotiation phase. In many situations, the consumer will initiate negotiations with a selected provider. However, if a provider has had past interactions with a certain consumer and knows the consumers general service requirements, then the provider sends an unsolicited offer to the consumer. The consumer may either ignore or respond to this offer, thus entering the negotiation phase. The ability of either agent, regardless of role, to initiate these phases is not often supported in negotiation protocols (see Section 3.6.2).

3.5.1 Overview of Negotiation Tools

The WSAN Service provides the necessary tools for inter-agent negotiation. Through this service, agents access a structured interface, including objects and methods, that facilitates the negotiation process.

Table 3.3 provides an overview of relevant negotiation objects. Two important objects are *SessionInfo* and *NegotiationPolicy*. The SessionInfo object allows agents to store, organize and analyze the 'tree' of offers and counter-offers (see Figure 3.7). Analysis of the SessionInfo object allows agents to learn the preferences of the counter-agent, discover trends or patterns in the negotiation process and, in turn, improve negotiation strategies. The NegotiationPolicy object provides agents the core components to create, evaluate and respond to negotiation offers. Each policy is unique to its agent and contains the specific preferences (e.g. utility functions) which guide offer evaluation and (counter) offer creation.

Table 3.4 presents an overview of relevant methods. Methods are categorized by functionality. *Complimentary Assistance* offers tools for before and after the negotiation session, including a method for selecting a unique session identifier and storing the accepted Agreement. *Directory Assistance* offers providers the ability to advertise services via a publicly available service directory. Methods are also provided for consumers to find needed services based on related keyword searches. For instance, a consumer searching for "energy" will find a providers offering "power" or "electricity" as well. *Message Assistance* contains methods for receiving both negotiation messages as well as regular messages. A regular message is a message that is not part of the negotiation protocol. Out-of-bounds messages (e.g. agent exiting the marketplace) are sent via this channel.

The primary negotiation methods are grouped in the *Negotiation Assistance*. This group provides methods to request a specific Template and process such a request (e.g. find and return). This group also includes methods to create and send negotiation offers and counter-offers. Finally, this group includes

Table 3.3: An overview of relevant negotiation objects.

SessionInfo	Stores relevant information regarding a single negotiation session, such as the session identifier, agents, roles, negotiation state(s) and a message (e.g. offers) archive. Agents analyze this archive of past interactions to improve negotiation strategies.
NegotiationPolicy	Encapsulates the preferences of each individual agent; unique to each agent and domain dependent. Enables agents to evaluate offers using personal utility functions.
NegotiationMessage	Encapsulates various negotiation message types, including (1) template request, (2) template response, (3) negotiation offer, (4) agreement offer and (5) agreement. Includes relevant metadata, such as agents and the session identifier.
Template	A document advertising available services.
NegotiationOffer	A document requesting a particular configuration (e.g. price, quality) of a given service. Serves as offer and counter-offer during multiround negotiation.
AgreementOffer	A document requesting a particular configuration of a given service. This document is the output of an acceptable negotiation offer and the input for agreement creation.
Agreement	A document representing a successful agreement (e.g. SLA).

methods for the Agreement Layer, including sending an agreement offer and an accepted agreement.

Providers use the *Template Assistance* group to create and organize multiple Templates. In addition, methods are available to validate incoming offers against respective creation constraints. For example, a selected value for a service term must be within the range specified in the respective Template.

3.5.2 Negotiation Modes

The WSAN Service support two modes of negotiation: (1) *independent* and (2) *mediated*. Independent mode allows agents to communicate directly with one another using the provided objects and following the specified protocol. Agent communication with the WSAN Service is limited to the Complementary Assistance methods (see Table 3.4). In mediated mode, all agent communication is mediated by the WSAN Service. Essentially, all messages are sent to the WSAN Service and are then forwarded to the respective agent. The WSAN

Table 3.4: An overview of relevant negotiation methods.

Complimentary Assistance	Collection of methods to assist pre- and post-negotiation.
requestNewSession	Request a unique session identifier from the WSAN Service.
storeActiveAgreement	Store an accepted Agreement with the WSAN Service for auditing and monitoring purposes.
Directory Assistance	Collection of methods to assist discovery of service offerings.
publishTemplates	Publish keywords describing available services to a publicly accessible service directory.
searchTemplates	Search a publicly accessible service directory for all providers with Templates matching given keywords (e.g. energy, electricity).
Message Assistance	Collection of methods to assist communication between agents.
getNegotiationMessage	Retrieve an incoming negotiation message from the message queue.
getRegularMessage	Retrieve an incoming message outside of the fixed negotiation protocol.
Negotiation Assistance	Collection of methods to assist the Negotiation Layer.
processTemplateRequest	Search and return Templates matching the supplied keywords.
requestTemplate	Request all Templates matching given keywords from a specific provider.
sendAgreement	Send an acceptable or accepted Agreement in the Agreement Layer.
sendAgreementOffer	Send an offer in the Agreement Layer.
sendNegotiationOffer	Send a (counter) offer in the Negotiation Layer.
Template Assistance	Collection of methods to assist providers with Template storage, organization and retrieval.
validateOffer	Validate an agreement- or negotiation offer against the creation constraints of the related Template.

Service stores a copy of each message but performs no additional analyses on the content.

Each mode offers several advantages. In terms of scalability, independent mode reduces communication load on a single WSAN Service. This mode allows a given marketplace to support an increased number of negotiating agents, compared to mediated mode. Agents use mediated mode if additional assurance, monitoring or auditing is required. Centralizing and recording communication history lends itself to monitoring and auditing in the case of disputes between agents.

3.5.3 Experimental Validation

Several experiments measure communication, CPU overhead and scalability of the negotiation implementation. The first set of experiments measures communication and CPU overhead with a limited number of agents. The second set of experiments examines the scalability with an increased number of agents.

Note that these experiments show relative communication and CPU load for agents negotiating with the WSAN Service. However, these experiments do not attempt to define what level of load is acceptable or unacceptable. This is a subjective threshold that is highly dependent on the context and application.

The experiments execute on a single machine. The machine is a SUN SPARC Enterprise T5240 with 2 multicore 1.2GHz CPUs offering 128 hardware threads and 64GB of RAM. This machine runs Solaris 10 and AgentScape middleware with 1 AgentScape Location.

3.5.3.1 Description of Experiments

In total, 15 experiments investigate the CPU and communication overhead of the WSAN Service. Consumers and provider agents negotiate energy services[10]. Each experiment runs with a number of providers and a number of consumers. Table 3.5 lists the combinations of providers and consumers. The first experiment runs 1 provider and 1 consumer. The second experiment runs 1 provider and 10 consumers. Each experiment runs with a unique combination of providers and consumers. The final experiment runs 300 providers and 300 consumers.

All experiments proceed as follows. First, the experiment creates a single AgentScape Location. Then, this Location loads the provider agents. Providers publish Templates to the Directory Service of the Location. All

[10]Figure 5.2 in Chapter 5 gives the details of the energy services.

Table 3.5: Overview of WSAN Service experiments.

Providers	Consumers
1	1, 10, 100, 200, 300
10	10, 100, 200, 300
100	100, 200, 300
200	200, 300
300	300

Templates offer identical services and providers follow identical negotiation policies. Then, the Location loads the consumer agents. All consumers select a random Template from the Directory Service and begin negotiations with their respective provider. All consumers follow identical negotiation policies. After creating an agreement, a consumer waits for a period of time. This period of time is chosen randomly between a minimum of 20 seconds and a maximum of 60 seconds. After this period, a consumer randomly selects a new Template and the process repeats. After loading all agents, the experiment logs CPU and message overhead every second for a period of 1 hour of continuous negotiation. The average CPU load is computed.

3.5.3.2 Experimental Results

Figure 3.16 shows the experiments with 1 or 10 providers; 9 experiments in total. These results indicate the average percentage CPU load of negotiation between providers and consumers. A single provider and a single consumer (2 agents in total) generate 0.09% CPU load. A single provider and 10 consumers (11 agents in total) generate 0.30% CPU load. A single provider and 100, 200 or 300 consumers generate approximately 0.90% CPU load. 10 providers and 10 consumers (20 agents in total) generate 0.31% CPU load. 10 providers and 100, 200 or 300 consumers generate 2.30%, 4.45% and 6.40% CPU load, respectively.

Figure 3.17 shows the experiments with 100, 200 or 300 providers; 6 experiments in total. These results indicate the average percentage CPU load of negotiation between providers and consumers. 100 providers and 100 consumers (200 agents in total) generate 2.40% CPU load. 100 providers and 200 or 300 consumers generate 4.79% and 6.79% CPU load, respectively. 200 providers and 200 or 300 consumers generate 5.00% and 7.54% CPU load,

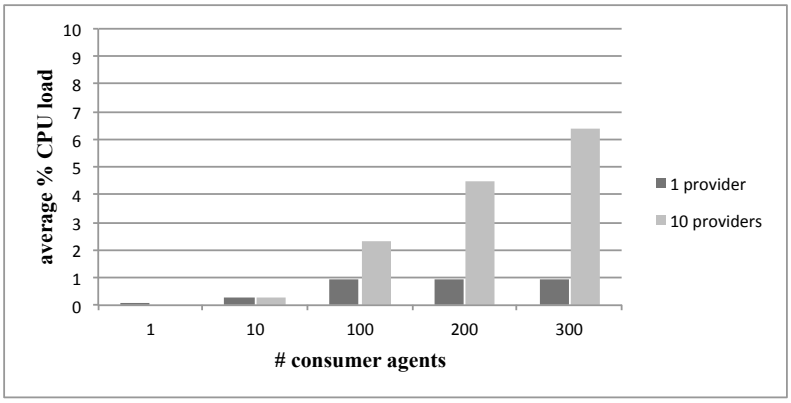

Figure 3.16: WSAN Service CPU load.

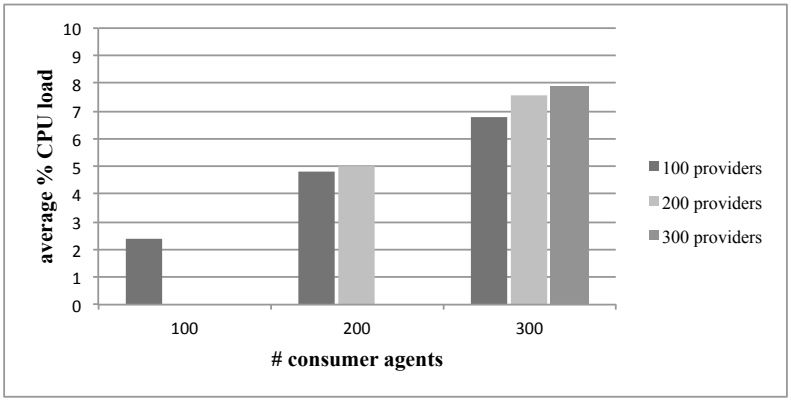

Figure 3.17: WSAN Service CPU load.

respectively. 300 providers and 300 consumers (600 agents in total) generate 7.92% CPU load.

Communication measurements show message count and message size. In this experimental setting (i.e. negotiation policy and strategy), a single negotiation session produces 8 messages. A single messages is approximately 2.5 kilobytes (kB). Therefore, a single negotiation session is approximately 20kB. This number does not change over time. If an agent negotiates 100 sessions in one hour, then total communication is 800 messages and approximately 2,000kB. This number is independent of the number of providers. For instance, 1 provider and 10 consumers produces the same communication overhead as 10 providers and 10 consumers. In this experimental setting, consumers initiate negotiation. Therefore, 10 consumers produce the same amount of overhead whether they all negotiate with a single provider or with 10 separate providers.

3.5.3.3 Discussion of Results

According to these experiments, the WSAN Service generates relatively low overhead. Even with 600 agents negotiating simultaneously, average CPU load is below 8%. In this experimental setting, CPU load is not affected by the total number of agents, but rather by the number of consumer agents. For this reason, CPU load is almost identical for 10 consumers and 1 provider or 10 providers. In both cases, the number of negotiation sessions (initiated by the consumer) remains the same, and thus the CPU load also remains the same.

These experiments show identical CPU load for 1 provider and 100, 200 or 300 consumers. This indicates that the provider is overloaded and can only process a certain number of the consumer messages within the given time. Adding more providers allows the system to scale. These experiments reveal that CPU load is not the main obstacle to scalability. In this case, the bottleneck preventing a provider from handling 200 or 300 consumers simultaneously lies elsewhere (e.g. disk I/0). Further experimentation is required to identify the exact bottleneck.

3.6 Related Work

The field of automated negotiation is large and active. The scope of this dissertation is limited to the area of multiagent negotiation of SLAs. Even this limited scope covers several alternative technologies and frameworks. Two specific technologies relevant to this research are SLA specification and negotiation protocols. A number of alternative approaches are available for each of these technologies, The following sections discuss these alternatives.

3.6.1 Agreement Specification

The Web Service Level Agreement (WSLA) specification [96] from IBM[11] is part of an extensive framework [81] for creating and monitoring SLAs in the web services domain. Tags are built into both the SDTs and GTs, using defined *metrics, functions, schedules, triggers, measurement directives* and *actions*. This framework defines five stages: (1) SLA negotiation and establishment; (2) SLA deployment; (3) Service level measurement and reporting initiated either by either the provider or the consumer; (4) Corrective management actions including notification, termination or prioritization of certain tasks, and (5) SLA termination. Steps 3 and 4 are the defining features of this approach and are not found in the WS-Agreement (`NPS-0`) or WS-Agreement Negotiation (`NPS-1`) specifications.

[11] International Business Machines: `http://www.research.ibm.com/`

XML describes Agreements with a structure similar to those defined by the WS-Agreement (NPS-0) specification. The three main elements are Participants, Service Description and Obligations. These broadly correspond to Context, Service Description Terms and Guarantee Terms of WS-Agreement. However, the WSLA Service Description element contains additional information regarding how services are measured and monitored This includes SLA Parameters such as service availability, throughput or response time, as well as, metrics and functions to compute these parameters. Each function also has a schedule that determines when it is computed.

In contrast to WS-Agreement (NPS-0), the Obligations element specifies guaranteed actions, in addition to service level objectives. This specifies the actions taken if a violation is detected, such as notifications or corrective actions. Among the corrective actions is the ability to prioritize tasks such that premium customers or time critical tasks are processed on time by pausing or canceling other tasks.

While WSLA offers an extensive library for specifying QoS and monitoring metrics, it offers no guidance for negotiation. No negotiation protocol is specified. Agents may use a negotiation protocol so long as documents use valid WSLA syntax.

Another approach to SLA specification is presented by [148, 149] from HP[12]. This specification offers an SLA specification language that is both precise and flexible. An XML structure is defined that contains many of the elements found in WS-Agreement, including the agents involved, the service terms and Service Level Objectives (SLO) (e.g. quality guarantees). In addition, the SLA contains *Service Level Indicators*, such as `availability`, `performance` and `reliability` with target levels to achieve. Furthermore, the SLA contains *Penalties* that explain exactly what happens if an agent is unable to meet the objectives in the SLA.

The SLO element contains a set of components to precisely explain how to test for SLA violation. For instance, these components specify measurements taken at the end of the month to test if availability was at least 99.9%. The components include: *measuredItem* - the set of data that is measurable; *evalWhen* - when the evaluation clause is triggered (or a fixed time); *evalOn* - sample selection and computation (e.g. the five longest periods or the average); *evalFunc* - the evaluation function applied to the sample set (e.g. a mathematical function), and *evalAction* - what action to take after the measurement.

As with WSLA, the specification from HP offers an extensive library of functions for measuring SLA compliance. In addition, more consideration is

[12]Hewlett Packard Laboratories. `http://www.hpl.hp.com/`

given to explicitly defining penalties in the event of violation. Also, as with WSLA, no negotiation protocol is specified.

3.6.2 Negotiation Protocol

The Service Negotiation and Acquisition Protocol (SNAP) offers a high-level overview of operations for SLA creation in distributed environments [43]. SLA documents are categorized as *Task Service Level Agreements* (TSLAs), *Resource Service Level Agreements* (RSLAs) and *Binding Service Level Agreements* (BSLAs). A TSLA specifies a specific task (e.g. compute job). An RSLA defines the resources or set of resources (e.g. 5GB RAM, 3.0 GHz CPU). A BSLA binds the task of a specific TSLA to the resources defined in a specific RSLA.

No specific document language or syntax is defined by SNAP. Rather, a high-level approach is presented for structuring SLA documents. A compositional language should describe resources as compositions of other resources. For example, a resource consists of hardware, software and network. Hardware includes CPU, RAM, DISK, etcetera. SLA documents also include specific resource metrics (e.g. time, max, min) to describe specific properties of each resource.

Resource offerings advertise available services. These offerings contain various constructions, such as *sets* or *alternatives*. A set is a combination of multiple resources that are all required together. In contrast, an alternative is a combination of multiple resources of which only one is required. These constructions correspond to `<ALL>` and `<CHOICE>` descriptors in WS-Agreement.

As in WS-Agreement Negotiation (`NPS-1`), this protocol explicitly allows renegotiation of existing SLAs. A client sends a new request for SLA that uses the existing unique identifier of the existing SLA. The issue of multiround negotiation capability is less clear. The authors stress the importance of multiphase negotiation as a tool for clients to explore the negotiation space and discover optimal resource configurations. However, the operations described by SNAP do not explicitly support multiround negotiation, in contrast to WS-Agreement Negotiation. The client makes a request and the provider accepts or rejects the request. The provider has no operations to explain the reason for rejection or propose an acceptable offer. It may be assumed that upon rejection, a client is permitted to alter the original request and try again.

In addition to the consumer and provider, a service manager is identified. This manager is responsible for complimentary services, such as creating unique identifiers and setting expiration times for each SLA. The role of the manager is somewhat similar to that of the WSAN Service (see Section 3.5).

SNAP assumes a trusted environment in which clients trust resource providers to act in good faith. As with WS-Agreement Negotiation, additional security mechanisms must be provided by the underlying architecture.

Another protocol for establishing agreements in distributed environments is the Contract Net protocol [159]. In contrast to WS-Agreement Negotiation (`NPS-1`), Contract Net is always initiated by a resource consumer, referred to in this protocol as a *manager*. The consumer advertises the specific task (e.g. compute job) to process and resource providers (referred to as *contractors*) submit bids to process the task. Then the consumer chooses the most acceptable bid. There is no support for multiround negotiation.

Agents use Contract Net to negotiate SLAs in Grid environments [120]. A multitiered SLA negotiation layer matches resources to tasks. Each consumer is presented with one or more resource bids from providers. A consumer selects one and the SLA is created. The protocol does not support multiround negotiation. Therefore, the consumer is not permitted to propose a different resource bid. The protocol does not support argumentation. Therefore, the consumer is unable to explain why a particular resource bid was rejected with the intention of improving future bids.

As discussed in Section 2.2.1, symmetry between roles is not a common design goal in this field. All of the work cited in this section use asymmetric roles. Roles are static and support different access to data or actions (i.e. methods). Managers (i.e. consumers) always initiate negotiation in the Contract Net protocol [159] and derivatives [57, 120]. SNAP also distinguishes between clients and resource owners. Different roles have different actions and access to data. In contrast, this dissertation presents extensions to the new WS-Agreement Negotiation (`NPS-1`) specification that support role symmetry. Agents have symmetrical negotiation data and actions (e.g. initiating negotiation, accepting an offer), regardless of role.

3.7 Conclusions

Automated negotiation requires well-defined protocols. Protocols provide structure within which software agents communicate (e.g. exchange negotiation offers). A protocol defines how agents communicate, what language they use and what actions they are able to perform. However, a negotiation protocol must also be flexible and allow agents to respond dynamically to environmental changes. Agents may change roles or preferences. Environmental components may change, such as prices, resources or even terminology.

The WS-Agreement (`NPS-0`) specification defines such a basic protocol for SLA creation. A clear language is used to describe service offerings and quality guarantees. However, this specification does not support multiple rounds of negotiation nor is the protocol symmetric. Symmetric protocols support flexibility, by allowing roles to change without requiring changes to an agent's permissions (e.g. data access) or abilities (e.g. initiate, accept).

This dissertation presents the new WS-Agreement Negotiation (`NPS-1`) specification. WS-Agreement Negotiation extends the previous specification with support for multiround negotiation with limited argumentation. With this new specification, agents enter a bidirectional, dialogue. If an offer is unacceptable, rather than terminating negotiation, an agent may respond by creating a counter-offer. In addition, the research presented in this dissertation contributed to this specification to make the protocol both stateless and asynchronous. These attributes make the protocol better suited to MAS. After rigorous review, WS-Agreement Negotiation (`NPS-1`) is an official OGF standard.

In addition, this dissertation proposes several extensions (`NPS-2`) to WS-Agreement Negotiation to (1) explicitly converge a negotiation session, (2) increase symmetry of roles and (3) clarify intervals. This specification is implemented and experimentally validated in AgentScape.

CHAPTER 4

Service Monitoring in Open Environments*

The previous chapter describes consumers and providers negotiating services. A successful negotiation results in the creation of a Service Level Agreement (SLA) (see Section 3.1.2). This document describes the negotiation participants involved (e.g. consumer and provider), the provisioned services and the specific quality guarantees of those services. An SLA also specifies what actions to take if a participant violates the agreement. For instance, a consumer negotiates access to hardware resources from an online provider. The resulting SLA specifies the price the consumer must pay as well as the exact description of the resources (e.g. disk space, RAM). Additionally, the SLA guarantees 99% uptime and network latency lower than 1 second. If these guarantees are violated, the SLA dictates a reduction in price of 10% per violation. Once the SLA is finalized, the consumer uploads and runs private software on the rented hardware.

It now becomes necessary for both the consumer and provider to know if, when and by whom the agreement is violated. If the consumer claims an SLA violation has occurred, the provider may lose revenue, unless the provider is able to show that no violation actually has occurred. To offer participants assurance that violations are detected and nonviolations are ignored, services are monitored. Relevant service metrics (e.g. response time) are measured at periodic intervals to offer assurance that a service is being provided as promised.

*This chapter is based on four published papers [29, 33, 34, 132].

Either a participant (e.g. consumer or provider) or a separate monitoring service performs these measurements.

If a participant suspects a violation has occurred, monitoring data are analyzed. This analysis reveals which, if any, specific service guarantee is violated and which participant (e.g. consumer or provider) is responsible. In the example above, the provider guarantees 99% uptime for its hardware. If monitoring data reveals that the service crashed and was offline more than 1% of the time, the provider is responsible for a violation of the SLA. However, the violation may also be the responsibility of the consumer. For instance, if monitoring data reveals that the crash was the result of a fault in the consumer's own software, the consumer is then responsible for the violation. In addition, there could also be an external reason for the failure (e.g. force majeur: a lightning strike disables the connecting communication lines). Such insight into violations and responsible participants requires service monitoring.

Service monitoring in open environments presents several challenges. One challenge is that of trust. Participants are not implicitly trustworthy. Participants may lie about the quality of the service they are either providing or receiving. Therefore, monitoring measurements must be performed objectively, and impartially. One possibility is to provide each participant with access to the other's internal monitoring sensors. However, a particular participant may not wish to give external participants access to local resources and (sensitive) data that are required to perform monitoring measurements. One possible solution to this conundrum is the use of a Trusted Third Party (TTP) as a separate monitoring service. The TTP offers additional assurance and performs measurements impartially so monitoring results are trustworthy.

An additional challenge in open environments is that of scalability. A monitoring solution must be able to scale across multiple machines and handle multiple participants, distributed across multiple administrative domains. Centralized approaches often become a processing bottleneck. Therefore, monitoring in open systems requires inherently decentralized solutions. Generally speaking, the transition from centralized to decentralized solutions is a trade-off of processing overhead for communication overhead. This trade-off must be taken into consideration when monitoring services.

Another challenge is that of dynamism. Open systems are subject to constant change: changing participants, roles, resources, availability and demand. A monitoring service must be able to (automatically) adapt to these changes. A participant's resource requirements may also change over time. The importance of certain resources may change as may a particular participant's reputation. These factors affect a participant's perceived level of risk. The level of perceived risk for a specific participant reflects the likelihood and impact of

violation of a particular service transaction. Changing levels of perceived risk should be reflected in monitoring policy.

This chapter proceeds as follows. First, an overview introduces the main concepts of distributed service monitoring, including traditional monitoring techniques. Secondly, this chapter describes an alternative monitoring technique known as *passive monitoring*. Thirdly, this chapter proposes self-adaptive monitoring that combines traditional and passive monitoring techniques. Fourthly, this chapter describes an implementation of this monitoring technique in the AgentScape middleware, including experimental results. Finally, the research in this chapter is compared to related work.

4.1 Introduction

This section provides an overview of the various aspects of service monitoring, starting with examples from literature of the traditional approach to service monitoring. The overview also highlights several drawbacks of this approach. A generic monitor design with an overview of monitoring components provides insight into the inner workings of service monitors.

After this general overview, this section discusses several specific aspects of service monitoring design. This includes (1) security and reliability; (2) distribution and decentralization, and (3) dynamism and adaptation. Security and reliability aspects address possible attacks on monitoring integrity and suggested countermeasures to prevent such attacks. One particular countermeasure is the use of a Trusted Third Party (TTP). A TTP performs monitoring measurements such that results remain objective and trustworthy.

Another important aspect of monitor design is distribution associated with open systems. As discussed in Section 1.1, open systems are large environments that are distributed across multiple hosts that span geographical and administrative borders. In such environments, highly centralized monitoring processes become bottlenecks that prevent adequate scaling. A successful service monitor is designed such that it scales and functions effectively in such environments.

Service monitors must also be capable of adapting to the highly dynamic nature of open environments. Factors of adaptation include the overhead created by the monitoring process, such as communication, processing and additional costs. Additional costs are associated with certain monitoring activities that require additional (hardware) resources or external (paid) services. Another factor of adaptation is that of perceived risk.

Following the discussion of monitor design, several additional aspects are presented regarding SLA enforcement. This includes (1) auditing and conflict mediation; (2) penalizing violations, and (3) policy specification. Conflict mediation is the process of determining whether or not an SLA violation has occurred, which specific term is violated and which participant (e.g. consumer or provider) is responsible. Once the responsible participant is identified, penalties are enforced following a defined policy. One approach to specifying policies is to negotiate and include them directly in the SLA. Policy information includes what penalties to enforce, how and when to enforce them.

4.1.1 Active Service Monitoring

Most traditional monitoring services periodically test various metrics at specified intervals (e.g. 5 seconds) to determine if a system is operating as expected. For instance, distributed computer systems often use "heartbeat" monitoring to detect if a node (i.e. a computer or component) has failed [67]. Nodes are periodically contacted and asked to respond. Failure to respond indicates that a node (or an intermediate network) has failed and corrective action is required. This mode of monitoring is sometimes referred to as online, continuous monitoring or active monitoring. For clarity, the remainder of this dissertation refers to this mode solely as *active monitoring*. Using this technique, an impartial monitoring service (e.g. TTP) takes measurements on behalf of the consumer and provider. If a violation is detected, the monitoring service takes action. Such action could be to cancel the service or penalize the offending participant. The chosen action depends on the policy agreed upon by the participants during service negotiation. Section 4.1.7 discusses penalties in more detail.

SLAs are also actively monitored [95, 132]. A service is monitored by periodically testing whether the terms of an SLA are violated. This may require measuring a single variable or a complex aggregation of variables. For instance, *'Host is reachable.'* may be measured by a single request/response action. In contrast, *'Host uptime is greater than 99%.'* is measured by polling a host multiple times and calculating the average rate of success.

If an SLA violation occurs, measurements collected with active monitoring identify the responsible participant. In some cases, monitoring data may also exonerate a participant suspected of violation. For instance, most SLAs assign no penalties in the case of force majeur (e.g. natural disaster). If a lightning strike disables the communication lines between a consumer and a provider, the consumer may incorrectly conclude that the provider is responsible for the SLA violation. Monitoring data shows that the provider is not responsible for this particular violation.

An external, impartial service performing active monitoring offers high assurance that SLA violations are detected and offending participants are identified. However, the assurance offered by active monitoring comes at a cost. This mode of monitoring may require substantial resources, including monetary, hard- and software. Furthermore, this mode of monitoring relies on an external service for its impartiality. External services add both additional complexity and additional costs.

One drawback of active monitoring is the difficulty and importance of choosing a measurement interval. The interval between measurements ranges from a few seconds to hourly or daily measurements, depending on the nature of an SLA. The accuracy of active monitoring depends directly on the chosen interval. The shorter the interval, the more accurate and comprehensive the results, and vice versa. For instance, if a sporadic service failure lasts for 30 seconds, testing every 5 seconds will be more likely to detect this failure than testing every 5 minutes. The chosen measurement interval also directly impacts processing and communication overhead. Overhead is a result of the tests, messages and analysis generated by a monitoring process. The shorter the interval, the higher the overhead, and vice versa. The choice of a specific measurement interval is therefore often a trade-off between accuracy and overhead. Many optimizations may lower overhead, such as dynamically decreasing the frequency [83] or lowering the complexity [110] of measurements. Section 4.1.5 discusses these optimizations in more detail.

Another drawback of active monitoring is dependence on an external service. In some cases, consumers and providers may not prefer to provide external parties (e.g. a TTP) with access to local resources and (sensitive) data. However, the TTP requires this access to perform accurate measurements. In other cases, dependence on a TTP forms an obstacle to scalability of a system. For instance, if a provider relies on a single TTP and this TTP becomes overloaded or suffers a severe failure, the provider is affected and is unable to provide monitoring for existing consumers or accept new consumers. A TTP may also incur additional costs thus making active monitoring more expensive in terms of monetary payment or additional overhead. Section 4.2 discusses an alternative monitoring technique that reduces dependence on a TTP.

4.1.2 A Generic Monitor Design

Two main conceptual components of a service monitor are *Monitor Sensors* and *Monitor Processes*. Monitor Sensors are positioned within strategic locations to measure services. These sensors must have direct access to local metrics of host machines of a provider, as well as a direct connection to a consumer's machine and all communication in between. Sensors are passive

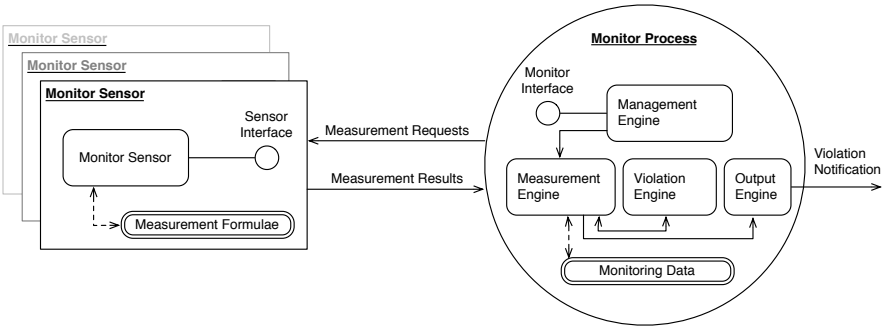

Figure 4.1: A generic monitor design.

in the regard that they should not take action or affect the local system until they receive a request from a Monitor Process. When a request is received, a sensor becomes active, performs a measurement, returns the results and then becomes inactive again.

The bulk of monitoring logic is defined in a separate Monitor Process. The responsibilities of the Monitor Process are: (1) identify which SLA terms require monitoring; (2) request measurements from Monitor Sensors; (3) check results for violations, and (4) take appropriate action if a violation is detected. Figure 4.1 illustrates the design of the Monitor Sensor and Monitor Process.

Each Monitor Sensor provides an interface for communication with a Monitor Process and a local library of measurement formulae. This contents of this library are standardized or otherwise agreed upon (e.g. during negotiation). A Monitor Sensor listens for measurement requests, retrieves measurement formulae from a local library, performs the measurement and returns the results.

A Monitor Process has an interface component and four engines: a Management Engine, a Measurement Engine, a Violation Engine and an Output Engine. An interface component receives new SLAs, sends measurement requests and receives results. A single Management Engine module coordinates and stores the SLAs, measurement results and recorded violations. This module creates a new Measurement Engine for each active SLA. The Measurement Engine checks if one or more of the SLA terms require monitoring and, where appropriate, begins monitoring these items by communicating with sensors. Results are tested for violations based on a (negotiated) violation policy. If a violation is detected, the Violation Engine is notified. This module takes action as specified by a violation policy, such as contacting the Output Engine to inform one or more participants.

Measurements may include static facts, such as the presence or absence of a required item (for example, a boolean value, such as '*host is reachable*').

These measurements also include dynamic items that require aggregated data to calculate the average, minimum, maximum or complex functions (for example, that return a real number, such as '*average uptime greater than 99%*'). If a violation is detected during active monitoring, action is taken, as specified by the SLA. When the SLA ends, the results are stored, along with the communications log, in encrypted form for possible later auditing or conflict mediation. Additional mechanisms are available to secure these data, such as append-only storage [179].

4.1.3 Security and Reliability

An important aspect of monitoring in open environments is safeguarding objectivity of monitoring results. A participant may decide to manipulate the results to its advantage. For instance, an SLA may stipulate that a consumer receives financial compensation if a specified service is unreachable. Regardless of the actual status of the service, that consumer may decide to manipulate monitoring results to make the service appear to be unreachable to collect financial compensation.

To provide reliable measurements, the monitoring process must therefore be secure against malicious participants that attempt to violate agreements or interfere with the provision of services to others. Thus, the data collected from monitoring must be protected from deletion or modification by unauthorized participants. A secure logging mechanism [65] accurately records and securely preserve the record of communications and measurements. Protection of these data is needed as stored data also serves as an audit trail to detect violations offline, after service provisioning has ended.

Furthermore, monitoring should rely as little as possible on data provided by participants and should attempt to rely solely on independent measurements. The concern for independence determines the placement of a monitoring module. Rana et al. distinguish three possible locations for monitoring [140]:

Trusted Third Party (TTP) - an independent module that monitors (and logs) all communication between consumers and providers. Upon successful creation of an SLA, both participants receive a signed certificate from the TTP. This certificate serves as nonrepudiation and/or reputation building of the provider. However, a TTP is unable to measure the internal state of either a consumer or provider.

Trusted Monitoring Module (TMM) at provider - functionally equivalent to a TTP but with access to the internal state of the service provider. However, a provider may not reveal all of the internal state or may report incorrect information to the monitor. A module at this location may

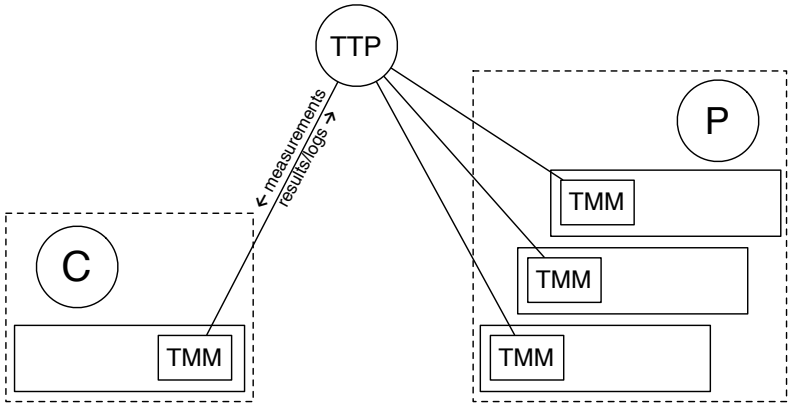

Figure 4.2: Combination of TTP and TMMs at consumer and provider locations.

prove that a provider attempted to avoid violations or took appropriate (i.e. following official protocol) action when they occurred.

Trusted Monitoring Module (TMM) on the consumer site - functionally equivalent to a TTP but it is difficult to distinguish between provider delay and network delay. A module at this location is not only useful for measurements, but also for establishing the trust level for certain providers.

Figure 4.2 depicts a combination of these possible locations. This combination results in both independence of measurements and access to the internal state of both provider and consumer. In this figure, a TMM is installed on each host within both consumer and provider locations. A TTP communicates directly with each TMM to request measurements and collect results/logs. Independence and objectivity of monitoring results is enforced by a Trusted Third Party (TTP) [33, 132] that performs, analyzes and stores monitoring measurements. To prevent participants from manipulating the measurement results collected at respective locations, a TTP installs Trusted Monitoring Modules (TMM) within each participant's location. A TMM performs measurements such that results are trustworthy. Results are objective because a TMM is designed to resist malicious influence from unauthorized participants. A TMM could use a secure hardware module (e.g. Trusted Computing Platform [127]) to ensure that measurement sensors and data are protected from tampering.

The use of TMMs allows participants to have equal access to the same service metrics. For instance, a consumer may not allow a provider to access sensitive client data directly. However, a TMM allows a TTP to access these data securely. Thus, the provider has 'indirect' access to the data via the TTP

and will be notified if it reveals SLA violations. Which TMMs are required to access which service metrics depends on a specific SLA.

The monitor should also be reliable and robust to system failure and overload. In some cases, distribution of the monitoring process supports both of these requirements. A distributed monitor removes the risk of a single point of failure. The monitoring process should be self-healing. Firstly, workload should be automatically balanced across monitors, preventing a single monitor from becoming overloaded. Secondly, monitors should automatically recover from failures and these failures should not affect the accuracy or integrity of the data that has already been collected.

4.1.4 Distributed and Decentralized

Another aspect of open environments is the challenge presented by distribution of participants and resources. As discussed in Section 1.1, open environments may be spread across many geographical areas and administrative domains. Participants from one domain or one area may access resources located far away and administered by a different organization.

Monitoring services in such environments requires that a monitoring process has certain characteristics. Firstly, the monitoring process must be able to access relevant service metrics regardless of geographical or administrative location. This is accomplished with the use of privileged sensors (e.g. a TMM). Secondly, the monitoring process must be able to scale as the environment grows with more participants joining the system from various locations. The monitor must continue to function even as processing and communication overhead increases.

Designing monitoring systems for distributed environments is an area of ongoing research. Grid and Cloud environments currently monitor a wide spectrum of metrics, from low-level hardware health to high-level service compliance [150]. Two well known monitoring systems are Nagios [112] and Ganglia [103]. These systems scale to monitor many system metrics distributed across many hosts. However, both systems are designed for use within a single administrative domain and rely on centralized components.

Centralization of components forms a possible roadblock that prevents systems from scaling in open environments [165]. Centralization may lead to bottlenecks in communication, processing or storage. For example, consider the waiting time if a single operator is responsible for personally answering all emergency telephone calls for an entire country. Several techniques are applied to reduce centralization, including replication or distribution of data and processing. For instance, if several, identical monitoring services are available, a consumer decides to choose the one with the least communication delay or

fewest current participants. Another option is to create monitoring protocols that reduce interaction with centralized components. Section 4.2 describes one such protocol.

Decentralization not only increases the scalability of a system but also makes the system more robust to failure. For instance, the failure of a certain monitoring component should not cause the entire system to become inoperable. A monitoring framework should be able to replace the failed component without loss or corruption of monitoring data.

4.1.5 Dynamic and Adaptive

Open environments are subject to constant change. Resources, attributes and availability, change over time. Participants join and leave systems. During interaction with a system, participant needs, preferences, attributes, reputation and even identity may change. Depending on the circumstances, a participant's monitoring needs may also change. Some service interactions require more assurance that guarantees are met and some interactions require less. A service monitor should be able to adapt to these changes.

One area of adaptation concerns monitoring overhead. Monitors may automatically reduce processing or communication overhead when possible. Some monitoring services are currently able to dynamically adapt monitoring policy at run-time based on environmental limits or changes in priorities. For instance, the monitor proposed in [83] collects system notifications from distributed nodes and dynamically adjusts the frequency of the notifications, based on CPU load. The higher the load (e.g. more participants in the system), the lower the frequency of notifications. [110] describes an adaptive system monitor. This monitor attempts to reduce monitoring overhead by pre-selecting and focusing on key metrics. Only when an anomaly is detected in one of these key metrics, does the monitor adapt by increasing the number of related metrics that are continuously monitored. Effectively, this monitor is able to 'zoom in and out' of areas when problems are detected.

Another area of adaptation concerns a participant's perceived risk. Perceived risk is a combination of factors including the importance of a particular service transaction (e.g. high financial value) and the perceived likelihood that something may go wrong. A monitor that adapts automatically to perceived risk offers participants a high level of monitoring assurance when participants deem necessary. The monitor then also reduces monitoring assurance and its associated costs (e.g. overhead or monetary charges) when participants deem permissible. Thus a balance is struck between high assurance and low costs. Section 4.3 discusses automatically adapting a monitor to the level of perceived risk.

4.1.6 Auditing and Conflict Mediation

The next several sections discuss SLA enforcement. Violation detection begins with auditing. Audit logs are kept, regardless of monitor design or which participant performs measurements. All relevant monitoring actions are logged, including measurement results and relevant communications between participants. Before service provisioning begins, participants must agree what data to store and the duration of storage. For example, store audit logs until all participants acknowledge that the SLA is successfully completed.

When a participant suspects that a violation has occurred, the participant requests conflict mediation. Mediating conflicts is essentially a process of auditing the monitoring logs to determine whether or not a violation has actually occurred and who, if anyone, is responsible. To prevent malicious manipulation of audit logs, several security measures are taken. These include secure logging mechanisms [65] and the use of a TTP.

The process of conflict mediation begins with the TTP collecting monitoring logs from all relevant hosts (e.g. client, server). Once collected, the TTP audits these logs by searching for results that violate the SLA. For instance, the TTP may discover measured response times in the logs that exceed a specific Service Level Objective (SLO). If possible, the TTP identifies the responsible participant and enforces penalties.

Conflict mediation may also reveal that no participants are responsible for a particular violation. For example, a provider contests its liability for failing to meet a given SLO. The SLO is violated; however, the cause of the violation is beyond the control of the provider. In this case, the violation was a result of force majeure (e.g. a lightning strike) or even a malicious, targeted attack by external forces, such as a Distributed Denial of Service (DDoS) [90, 106] attack.

It is also necessary to determine how violations are recorded. Violations are either stored in the TTP, added explicitly to the SLA document [123] or included in usage records [99]. SLA status updates are either pushed to participants at intervals or published to a secure site to allow participants on-demand access.

4.1.7 Penalizing Violations

When a violation occurs, often a penalty is incurred. Penalties may be as simple as terminating the current agreement and finding a different provider, or more complex reputation or monetary based penalties [133]. Trust and reputation systems commonly penalize violations in service provisioning [79]. In these systems, reputation is a community-wide metric of an participant's

trustworthiness. This metric increases if the a participant completes transactions without violating an agreement. Conversely, the metric decreases if a term is violated. Reputation based penalties utilize the notion that consumers prefer providers with a higher reputation and try to avoid providers with a lower reputation. In contrast, monetary based penalties operate on the assumption that consumers pay less for poor service and more for better service.

Both of these mechanisms require additional infrastructure and security measures [79]. A reputation based system requires a persistent record of all transactions, both successful and violated. A monetary based system requires a secure means of payment, whether in currency or credit, that has actual value to the participants of the system. Both of these approaches require a means of guaranteeing that identities are unique, persistent and legitimate. For instance, underlying authentication mechanisms verify that participants are indeed whom they claim to be.

One form of monetary-based penalty is the use of escrow. Essentially, an agreed upon amount is deposited by both participants at a TTP. In the event of violation, the deposit effectuates penalty payment. If no violation occurs, the deposits return to respective participants.

Penalty terms must be both extensive and clear. Both participants must agree upon several aspects, including what constitutes a violation, how violations are penalized and which specific TTP to use. The exact penalty terms may be separately negotiated during SLA negotiation, as describe in the next section, or follow known policies, such as the following [140]:

All-or-nothing provisioning - provisioning of a service must meet all SLOs. ALL of the SLO constraints MUST be met to satisfy the SLA;

Partial provisioning - provisioning of a service must meet some SLOs. SOME of the SLO constraints MUST be met to satisfy the SLA;

Weighted Partial provisioning - provision of a service meets SLOs that have a weighting GREATER THAN a [participant specified] threshold.

Negotiation of the violation policy is also required to determine, for example, the severity of a violation and appropriate action using the policies introduced above. The violation policy may be negotiated as a separate SDT during the negotiation phase. Policies are explained in more detail in Section 4.1.8.

While current work focuses on penalizing violations of SLAs, one alternative approach is to renegotiate the SLA during enactment. For example, such an approach would allow the producer and consumer to alter the SLA towards providing a more realistic deadline for the consumer and potentially reducing penalties that the producer would otherwise incur. Such a mechanism could

```
<ServiceDescriptionTerm ServiceName="SLAMonitoring">
   <Policies agen:type="policy">
     <ViolationPolicy xs:type="string">ALLORNOTHING</ViolationPolicy>
     <ViolationCount xs:type="int">3</ViolationCount>
     <ViolationAction xs:type="string">NOTIFY</ViolationAction>
   </Policies>
</ServiceDescriptionTerm>
```

Figure 4.3: Example monitoring policy Service Description Term.

take advantage of multiround sessions that are part of the original negotiation. For example, if a previous round had a longer deadline, at a lower price, the renegotiation might entail that both participants agree to select this SLA session as a replacement. This, however, would require both participants to store the entire set of negotiation sessions until after provisioning has been completed.

4.1.8 Policy Specification

The existing WS-Agreement specification (NPS-0) [5] contains a *BusinessValueList* to express the importance of a certain SLO. The BusinessValueList either declares a value explicitly or implies a value through a penalty or reward type. For instance, a penalty or reward type may hold a monetary value that indicates the importance of the SLO. Although this enables a basic mechanism for punishing poor performance and rewarding good performance, it is possible to use a richer and more flexible method to specify violation policies. As opposed to explicitly extending the WS-Agreement standard, it is possible to add a separate SDT to the SLA to specify these policies. This includes the ability to choose a violation policy, such as those mentioned in Section 4.1.7, as well as the number of acceptable violations and the actions that to taken.

Figure 4.3 illustrates an example violation policy. The value of *ViolationPolicy* specifies the violation policy and takes the value of: *none*, *allornothing*, *partial* or *weighted*. The value of *ViolationCount* specifies how many violations are detected before action is taken. This must be a positive integer. A higher number allows the provider a chance to detect and correct the service disruption before being penalized. The value of *ViolationAction* specifies the action taken when a violation is detected and takes the value of: *none*, *notify*, *penalize* or *cancel*. This allows for no action, notification of the participants, penalty enforcement (as specified by existing penalty clause) or cancellation of the service. Figure 4.3 illustrates an example in which the participants will be notified after 3 violations to an SLO.

4.1.9 Conclusion

Designing monitors for open environments presents several challenges, including security considerations, obstacles to scalability and a high level of dynamism. Security threats are substantially mitigated by the use of a TTP and TMM. Decentralization of the monitoring process allows monitors to scale in large scale, distributed environments. It is possible to design a monitor to adapt to changes in the environment, such as overhead or the level of perceived risk.

Several important aspects of SLA enforcement include maintaining audit logs, mediating conflict and penalizing violations according to participant specified policy. A monitor must record measurements in logs that are audited during conflict mediation to identify violations and responsible participants. A TTP performs conflict mediation to safeguard the objectivity of the mediation process. If violations are detected, the responsible participant is penalized according to policy. The specific policy terms must be agreed upon before service provisioning begins.

Traditional active monitoring techniques have several drawbacks, including overhead, external dependencies and centralized processes. Additional decentralization and reduction of external dependencies is achieved with an alternative monitoring technique. The following section presents this technique.

4.2 Passive Service Monitoring

An alternative to active monitoring (see Section 4.1.1) is *passive monitoring* [84]. One advantage of this technique is reduced dependence on an external monitoring service or Trusted Third Party (TTP). The assurance provided by the TTP during active mode is provided in passive mode by the Service Evidential Protocol (SEP). Essentially, passive monitoring uses cryptographic primitives to build a secure, nonrepudiable audit log. Each participant must simultaneously commit before adding an entry to the audit log. For instance, all participants must be satisfied with the current level of service before an entry is added. However, if one participant is not satisfied, no entry is added. Therefore, this protocol is also referred to as a *mutual commit* monitoring protocol. This section contains a general overview of the protocol.

The protocol is separated into two phases: data collection and conflict mediation. During data collection, each participant is responsible for local service measurements and does not use external monitoring service, such as a Trusted Third Party (TTP). Service provisioning is divided into discrete intervals. For provisioning to continue for the next interval, all participants of the

agreement must agree that they are thus far satisfied with the service and have not detected violations. Once all participants agree, a token (e.g. password) is exchanged. Using cryptographic primitives, the token is encrypted and signed to ensure that, once sent and received, no participant is able to deny sending or receiving the token. These tokens are aggregated using other cryptographic primitives to form an audit log of compliance.

The data collection phase uses cryptographic primitives to ensure confidentiality and nonrepudiation of messages. Messages are encrypted using asymmetric (e.g. RSA [144]) encryption to ensure that messages are only read by intended recipients. Messages are signed using digital signatures. Digital signatures mathematically prove the authenticity and integrity of a single message. A valid signature indicates that a message has actually been created by the sender and has not been modified en route. The inverse is also true: if a message bears the digital signature of a given participant, that participant is unable to deny or repudiate sending that particular message. This property is referred to as *nonrepudiation*. An aggregate signature is a specific type of digital signature that combines multiple signatures into a single signature [15]. In contrast to a normal digital signature, an aggregate signature can verify a collection of multiple messages in a single operation. Section 4.2.1 discusses the data collection phase and the cryptography involved in more detail.

If no violations are detected, no intervention is needed. Tokens are exchanged directly between the participants, so no external monitoring services are required. This reduces the costs associated with interaction with a separate monitoring service. However, if a participant suspects a violation has occurred, conflict mediation is performed. There are several approaches to conflict mediation. In the simplest approach, the service is immediately canceled. Another approach is for participants to work together to examine the audit log to determine which participant is responsible for the violation. Optionally, a TTP also examines the audit log. Section 4.2.2 describes the process of conflict mediation in more detail.

The passive monitoring protocol presented in this dissertation is a modified version of the protocol introduced in [84]. Section 4.2.3 describes the modifications of the protocol. This section also explains why modifications are necessary and what benefits they offer.

4.2.1 Data Collection

The data collection phase of passive monitoring uses the Service Evidential Protocol (SEP). This protocol builds a distributed audit log that serves as evidence as to whether or not a service was correctly provided. This passive monitoring approach uses several cryptographic primitives to collect evidence

$E(M, pk_{i(RSA)}) = C$	**Encrypt:** Using the message M and the public key $pk_{i(RSA)}$ as input, create the cipher text C.
$D(C, pk^{-1}_{i(RSA)}) = M$	**Decrypt:** Using the cipher text C and the private key $pk^{-1}_{i(RSA)}$ as input, create the original message M.
$S(M, pk^{-1}_{i(BLS)}) = \sigma_i$	**Sign:** Using the cipher text C and the private key $pk^{-1}_{i(BLS)}$ as input, create the signature σ_i.
$V(\sigma_i, C, pk_{i(BLS)})$	**Verify:** Using the public key $pk_{i(BLS)}$, the cipher text C and the signature σ_i as input, verify the signature.
$A(\sigma_1, \cdots, \sigma_n) = \Sigma$	**Aggregate Signature:** Having signatures $\sigma_1, \cdots, \sigma_n$ as inputs, create one short, aggregate signature.
$R(\sigma, pk_{1(BLS)}, \cdots, pk_{n(BLS)}, C_1, \cdots, C_n)$	**Verify Aggregation:** Having several public keys $pk_{1(BLS)}, \cdots, pk_{n(BLS)}$, several cipher texts $C_1, \cdots, C_n$, the related signatures, $[\sigma_1, \cdots, \sigma_n]$ and one aggregate signature Σ, verify the signature.

Table 4.1: Service Evidential Protocol functions.

of SLA compliance or violation. These primitives are: (1) asymmetric key cryptography for encryption and decryption of messages (e.g. RSA [144]), (2) a signing scheme to sign and verify messages (e.g. BLS [15]) and (3) aggregation of signatures for verification of multiple signatures. Table 4.1 lists the functions of SEP.

SEP uses a mediator (e.g. TTP) to resolve conflicts, such as SLA violation. A TTP has a pair of keys for asymmetric cryptography: $pk_{ttp(RSA)}, pk^{-1}_{ttp(RSA)}$. Section 4.3.4 describes conflict mediation in more detail.

Essentially, a single iteration of the protocol consists of the same message being passed back and forth four times between the consumer and provider. Each time a message is sent, it is encrypted or signed with a different key. The encryption and signature scheme provides the required security. These messages are added to an audit log that is later analyzed to prove that a certain message was indeed received by a certain participant. Figure 4.4 provides

a high-level overview of the protocol. Figure 4.5 shows the actions and computations of each step. The remainder of this section explains each step in detail.

Step 1 - A provider encrypts a message (e.g. the access code) with the TTP's encryption key to create a cipher text. The provider then signs the cipher text. The resulting signature is added to the aggregate signature.

Step 2 - The provider sends both the signature and the cipher text to the consumer.

Step 3 - A consumer receives, but is unable to directly decrypt the cipher text. First, the signature is verified to ensure that it came from the provider and was not altered in transit. The consumer then signs the cipher text using its signing key. Both the resulting signature and the received signature are added to the aggregate signature.

Step 4 - The consumer sends the new signature back to the provider as "proof of receipt."

Step 5 - The provider receives and verifies the signature. The provider then encrypts the original message again, but now uses the consumer's encryption key. The provider signs this new cipher text. Both the resulting signature and the received signature are added to the aggregate signature.

Step 6 - The provider sends both the new signature and the new cipher text to the consumer.

Step 7 - The consumer receives and verifies the signature. The consumer then decrypts the cipher text to reveal the original message. The consumer then signs the cipher. Both the resulting signature and the received signature are added to the aggregate signature.

Step 8 - The new signature is sent back to the provider.

Step 9 - The provider receives and verifies this last signature and adds it to the aggregate signature.

At this point in the protocol, both participants have equivalent aggregate signatures Σ_p and Σ_c. These signatures are can only be created if a participant has access to the original messages, in the original order. Possession of an aggregate signature implies possession of the original messages. In addition, all participants are able to cryptographically show that all messages have been received and contents have not been manipulated.

4.2.2 Conflict Mediation

When conflict mediation is requested, the mediator (e.g. TTP) uses the data collected by each of the participants individually using the Service Evidential Protocol (SEP). The TTP investigates these data to determine if an SLA has been violated. To this purpose, the TTP first requests all data that has been

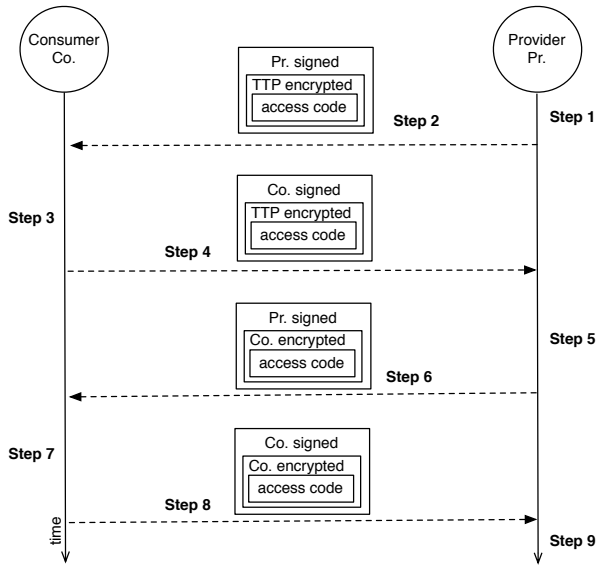

Figure 4.4: High-level overview of Service Evidential Protocol.

collected by each participant (i.e. signatures and encrypted messages). The TTP then performs cryptographic checks on the data to verify its integrity (i.e. the data has not been maliciously tampered with). The TTP then determines which, if any, participant has violated the SLA. This protocol is a modified version of the protocol introduced in [84].

Essentially, the TTP collects the audit logs from both the consumer and provider. The TTP then checks if these audit logs are equal or have been falsified or modified. If a modified audit log is detected, that participant is penalized. If both logs pass inspection, the TTP assumes that there was an error in communication, such as a lost message. The TTP generates and resends these missing messages. Figure 4.6 details the steps of this protocol.

Step 1 - First, the TTP requests the aggregate signatures and collections of signatures from each participant. In addition, the TTP requests the received messages (cipher texts) from the consumer.

Step 2 - Given this information, the TTP then tests the validity of the first cipher text and corresponding signature. If verification fails, the provider has provided a false message (e.g. access code). The protocol aborts and the provider is penalized.

Step 3 - The TTP then tests the validity of the first cipher text and the corresponding receipt signature. If verification fails, the consumer has provided a false signature. The protocol aborts and the consumer is penalized.

Step 1	Pr:	$C_1 = E(M, pk_{ttp(RSA)})$ $\sigma_{p1} = S(C_1, pk^{-1}_{p(BLS)})$ $\Sigma_p = A(\sigma_{p1})$
Step 2	Pr → Co:	$\texttt{message}\{C_1, \sigma_{p1}\}$
Step 3	Co:	$V(\sigma_{p1}, C_1, pk_{p(BLS)}) = \{pass, fail\}$ $\sigma_{c1} = S(C_1, pk^{-1}_{c(BLS)})$ $\Sigma_c = A(\sigma_{p1}, \sigma_{c1})$
Step 4	Co → Pr:	$\texttt{message}\{\sigma_{c1}\}$
Step 5	Pr:	$V(\sigma_{c1}, C_1, pk_{c(BLS)}) = \{pass, fail\}$ $C_2 = E(M, pk_{c(RSA)})$ $\sigma_{p2} = S(C_2, pk^{-1}_{p(BLS)})$ $\Sigma_p = A(\sigma_{p1}, \sigma_{c1}, \sigma_{p2})$
Step 6	Pr → Co:	$\texttt{message}\{C_2, \sigma_{p2}\}$
Step 7	Co:	$V(\sigma_{p2}, C_2, pk_{p(BLS)}) = \{pass, fail\}$ $D(C_2, sk_b) = M$ $\sigma_{c2} = S(C_2, pk^{-1}_{c(BLS)})$ $\Sigma_c = A(\sigma_{p1}, \sigma_{c1}, \sigma_{p2}, \sigma_{c2})$
Step 8	Co → Pr:	$\texttt{message}\{\sigma_{c2}\}$
Step 9	Pr:	$V(\sigma_{c2}, C_1, pk_{c(BLS)}) = \{pass, fail\}$ $\Sigma_p = A(\sigma_{p1}, \sigma_{c1}, \sigma_{p2}, \sigma_{c2})$

Figure 4.5: Single iteration of Service Evidential Protocol. Adapted from [84].

Step 4 - If both verification steps succeed, the TTP continues to recompute and test if the original messages are equal. The provider could have sent a valid first message, encrypted with the TTP's key. The provider could have then sent a false second message, encrypted with the consumer's key. To detect this, the TTP must compare the contents of the first and second cipher text. As the TTP does not have access to the decryption key of the consumer, the messages are compared as follows: First, the TTP decrypts the first cipher text using its own decryption key. Then, the TTP encrypts this message using the consumer's encryption key. The TTP has now effectively recreated the second cipher text without requiring access to the consumer's encryption key. If this newly created cipher text is not identical to the second cipher text provided by the consumer, the provider has sent a false message. The protocol aborts and the provider is penalized.

Step 1	Co → TTP:	`message`$\{C_1, C_2, \Sigma_c, [\sigma_{c1} \cdots \sigma_{cn}], [\sigma_{p1} \cdots \sigma_{pn}]\}$
	Pr → TTP:	`message`$\{\Sigma_p, [\sigma_{c1} \cdots \sigma_{cn}], [\sigma_{p1} \cdots \sigma_{pn}]\}$
Step 2	TTP:	IF $[V(\sigma_{p1}, C_1, pk_{p(BLS)}) = fail]$ THEN `abort protocol and penalize Pr`
Step 3	TTP:	IF $[\ V(\sigma_c, C_1, pk_{c(BLS)}) = fail\ ]$ THEN `abort protocol and penalize Co`
Step 4	TTP:	$\bar{M} = D(C_1, pk^{-1}_{ttp(RSA)})$ IF $[E(\bar{M}, pk_{c(RSA)}) \neq C_2]$ THEN `abort protocol and penalize Pr`
Step 5	TTP:	IF $[R(\Sigma_p, pk_{1(BLS)}, \cdots, pk_{n(BLS)}, C_1, ..., C_n) = fail]$ THEN `abort protocol and penalize Pr`
Step 6	TTP:	IF $[R(\Sigma_c, pk_{1(BLS)}, \cdots, pk_{n(BLS)}, C_1, ..., C_n) = fail]$ THEN `abort protocol and penalize Co`
Step 7	TTP → Pr:	`message`$\{\sigma_c\}$
	TTP → Co:	`message`$\{C_2\}$
Step 8	Pr:	$V(\sigma_c, C_1, pk_{c(BLS)}) = \{pass, fail\}$
Step 9	Co:	$D(C_2, pk^{-1}_{c(RSA)}) = M$

Figure 4.6: Conflict mediation protocol. Adapted from [84].

Step 5 - If the cipher texts are identical, the TTP proceeds to verify the aggregate signatures provided by each participant. If verification fails, the protocol is aborted and the responsible participant is penalized.

Step 6 - If verification succeeds, the TTP concludes that no one has (intentionally) violated the SLA. For example, perhaps messages were lost or corrupted in transit. The TTP resends the second cipher text to the consumer and the corresponding signature to the provider and the service continues.

Step 7 - The provider receives and verifies the final signature.

Step 8 - The consumer receives and decrypts the final message and is now able to access the service.

4.2.3 Protocol Modification

This version of the protocol is slightly modified from the original, found in [84]. In the original protocol, the second cipher text (C_2 in Step 6 in Figure 4.5) was never signed by either participant, nor was the signature returned to the provider. This made the protocol susceptible to two attacks: (1) the provider

could provide a "fake" access code and (2) the consumer could falsely deny receiving a correct access code.

In the first case, the attack would work as follows. The provider sends the correct access code in the first message, signed by the public encryption key of the mediator (e.g. TTP). Receipt of this message is acknowledged by the consumer with a new signature. The provider then generates a new, "fake" access code and encrypts this with the public encryption key of the consumer. This is then sent to the consumer, who is then unable to access the service. Conflict mediation is requested but the mediator is only able to decrypt and test the first access code, which is correct and thus is unable to prove that the second access code was "fake." The provider may falsely claim that the service was provided correctly and thus is not subject to penalty.

In the second case, the attack works as follows. The consumer receives the second message containing the correct access code, but claims that it was a "fake" code and requests mediation. The mediator is again only able to test the first message and is unable to make claims about the authenticity of the second message. The consumer may falsely claim that the provider has violated the agreement and withhold payment or break a (long term) contract without penalty.

In both cases, the apparent flaw in the original protocol is that without the second signature (σ_{c2} in Step 8 in Figure 4.5), the two messages C_1 and C_2 are not linked. Furthermore, during mediation, the mediator is only able to test the first message and thus is unable to verify that the two codes are equal.

By adding the second signature, both participants are protected from these two attacks. If conflict mediation is requested, the mediator verifies the second message and corresponding signature to prove that this was indeed the message sent by the provider and received by the consumer. This prevents the second attack. Once verified, the mediator then decrypts the first message and re-encrypt it using the public encryption key of the consumer ($\bar{M}$ in Step 4 in Figure 4.6). At this point, the mediator compares the first and second messages sent by the provider. This prevents the first attack.

4.3 Self-adaptive Service Monitoring

As discussed in Section 4.1.5, a monitor should automatically adapt to fit a participant's requirements. This section presents the self-adaptive monitoring approach that attempts to combine the benefits of active (see Section 4.1.1) and passive (see Section 4.2) monitoring modes. This offers high assurance when needed and reduces overhead when possible. This approach is designed

around the notion that the level of monitoring required for a given service transaction is a reflection of *perceived risk*. For instance, the importance of detecting a violation in a mission critical service differs from that of detecting a violation in a nonmission critical service. Violations ultimately have different levels of impact (e.g. financial, operational).

The level of perceived risk of a service is the product of many factors. These factors include level of trust between participants, likelihood of violation and impact of violation. In this context, trust is defined as a combination of several metrics in electronic markets [72, 102, 163]. These metrics include personal metrics, such as transaction history, transaction cost and ability to verify results. These also include community metrics, such as popularity, activity and position of a participant in a community. These metrics change over time. For instance, a successful transaction 10 years ago has less impact on the level of trust than a successful transaction yesterday.

A participant's reputation is an important factor in determining levels of trust. For instance, reputation often determines the first service provider a consumer chooses [143]. If a provider is well-known (e.g. Amazon Web Services, Ebay), this increases a consumer's initial trust. Once service provisioning has begun, a consumer dynamically adjusts his/her level of trust based on factors such as the number of successful transactions completed.

Different services have different levels of perceived risk and, therefore, require different levels of assurance with monitoring. The 'amount' or 'level' of monitoring may need to adapt accordingly. The term *Self-adaptive monitoring* describes this adaptation.

4.3.1 Adaptation Model

The basic building blocks of the self-adaptive monitor are (1) modes (active and passive) and (2) intervals. Active mode offers higher assurance than passive mode due to the continuous interaction with a TTP. Passive mode offers reduced dependence on a TTP, thus reducing the associated costs (e.g. financial payments). Intervals may be reduced or increased. Regardless of the chosen monitoring mode, shorter intervals (e.g. 10 seconds) offer higher assurance than longer intervals (e.g. 1 day). Measurements at shorter intervals are more likely to detect failures that occur sporadically.

The self-adaptive monitor chooses between these building blocks to offer a monitoring configuration that best matches a participant's perceived level of risk. The higher the risk, the higher the level of monitoring and vice versa. In this dissertation, high level monitoring equates to frequent active measurements. In contrast, low level monitoring equates to less frequent active measurements or passive measurements.

The choice of which monitoring level to use for a particular service transaction or service period is based on a self-adaptive monitoring function. On the basis of a perceived *risk level (RL)* and a *monitoring policy (P)* this function chooses an appropriate monitoring level. The monitoring level is the combination of (1) a mode (active or passive) and (2) an interval (time between measurements). Each participant to an agreement performs two actions before a transaction or period of transactions. First, RL is calculated on the basis of the risk factors. Based on the RL, a participant then selects an appropriate monitoring level.

Self-adaptive monitoring requires that all participants use the same level of monitoring for a given transaction. If participants select different levels of monitoring, both participants ultimately use the highest level. For instance, if a service provider selects active mode and a consumer selects passive mode, then both participants use active mode. This guarantees that all participants have at least the minimum level of assurance required.

4.3.2 Risk Level

The self-adaptive monitoring function uses the current level of perceived risk. This level is calculated based partly on knowledge supplied by the service environment. The environment supplies information about a particular participant, including recent activity and popularity in the environment. An example of environmental knowledge is a reputation authority [163].

In addition to environmental knowledge, local knowledge also influences the level of perceived risk. This includes the price (or cost) of the current transaction and history, if any, of transactions with the given participant. These two factors influence levels of trust and perceived risk [102, 177, 178]. The *transaction cost* of a current transaction and the *transaction history* correspond to a particular level of perceived risk. In short, the higher the cost of a transaction, the higher the perceived risk. Conversely, the better the transaction history, the lower the perceived risk. Figure 4.7 illustrates the relationship between transaction cost and transaction history.

Transaction cost is an artificial value that reflects the negative impact that would occur if a certain transaction were to fail (e.g. the other participant violates the agreement). This value is derived by first mapping levels of transaction cost to ranges of actual price. As an example, transactions below 100 euro may correspond to a cost of 1. Whereas transactions between 100 and 200 euro may correspond to a cost of 2 and so on. This mapping differs between participants. Each participant may have his/her own mapping, customized following individual policies.

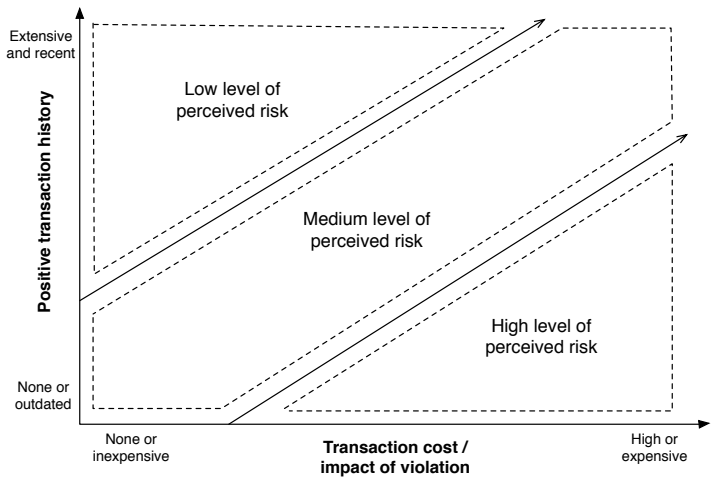

Figure 4.7: Relationship between local knowledge and perceived level of trust. Adapted from [102].

Transaction history is defined as a value that reflects the level of satisfaction with a given participant, based on past interactions with that participant. The higher the number of successful interactions in the past, the higher the transaction history. This value also takes into consideration the effect of *information decay*, as proposed in [163]. In this context, information decay means that recent transactions influence transaction history more than transactions that occurred long ago. Therefore, a weighting scheme assigns more weight (i.e. importance) to the outcome of the most recent transaction and less weight to older transactions.

4.3.3 Monitoring Policy

Each participant maintains a personal monitoring policy that specifies the relationship between a level of perceived risk and a level of monitoring. A monitoring level is defined as a monitoring mode (active or passive) and the frequency of measurements. For instance, a participant P decides that a risk level of 1 corresponds to a low level of monitoring. P thus defines a low level of monitoring as passive mode with an interval of 90 seconds. This participant decides that a risk level of 10 corresponds to a high level of monitoring. A high level of monitoring is then defined as active mode with an interval of 5 seconds. Each and every participant creates his/her own customized mapping between the level of perceived risk and the level of monitoring. A policy may change over time, based on lessons learned through interactions in the marketplace.

```
# context #
PolicyName=paranoid
ReactionThreshold=1

# age weighting #
10MostRecentTransactions=0.8
100MostRecentTransactions=0.1
...

# type weighting #
TransactionHistory=0.2
TransactionCost=0.8

# risk level mapping #
RiskLevel1Mode=passive
RiskLevel1Interval=90
...
RiskLevel10Mode=active
RiskLevel10Interval=5
```

(a) Paranoid

```
# context #
PolicyName=optimistic
ReactionThreshold=3

# age weighting #
10MostRecentTransactions=0.3
100MostRecentTransactions=0.5
...

# type weighting #
TransactionHistory=0.8
TransactionCost=0.2

# risk level mapping #
RiskLevel1Mode=passive
RiskLevel1Interval=90
...
RiskLevel10Mode=active
RiskLevel10Interval=20
```

(b) Optimistic

Figure 4.8: Examples of (a) paranoid and (b) optimistic monitoring policies.

In addition, a policy specifies that weights may be added to specific pieces of information. Weights indicate relative importance of information, based on age or type. For instance, the 10 most recent transactions may be twice as important as all past transactions. A policy also contains a value to indicate how quickly a monitoring process should react to changes in the level of perceived risk. This value prevents a monitoring process from constantly adapting to slight fluctuations in risk levels. A policy may contain additional metadata, such as a policy name.

Figures 4.8a and 4.8b illustrate two example policies: A *Paranoid* policy and an *Optimistic* policy. The paranoid policy has a low reaction threshold and thus quickly adapts its monitoring process to changes in the risk level. This policy also specifies a larger weight to the most recent 10 transactions and is more sensitive to transaction cost than history. In contrast, the optimistic policy has a higher reaction threshold and thus allows more variation in the perceived risk level before adapting the monitoring process. This policy balances the weight of most recent transactions with older transactions and considers history of transactions as more important than cost of a particular transaction.

4.3.4 Conflict Mediation

During service provisioning, if one of the participants suspects an SLA violation, conflict mediation is requested. Conflict mediation is handled by a Trusted Third Party (TTP). The actions taken by a TTP depend on the current monitoring mode. If the participants are currently using the active mode, the TTP consults the measurement results it has stored locally. If the participants are currently using the passive mode, the TTP performs a conflict mediation protocol. Section 4.2.2 describes the details of this protocol.

If no violation is detected, the service continues without change. If and when a violation is detected, the responsible participant is penalized. At this point, the participants decide how to proceed on the basis of the monitoring policies specified in the agreed SLA. For example, an SLA may dictate that the responsible participant is penalized with a monetary fine and the service continues, but the monitoring process is switched to active mode with a short measurement interval to reflect the heightened level of perceived risk, as proposed in the example in Section 4.3.5.

Figure 4.9 shows the possible state transitions when conflict mediation is requested. Monitoring starts either in active or passive mode. Monitoring policies dictate when to switch between modes (e.g. after a period without violation). If no violations occur, monitoring finishes normally when the agreement expires. If a suspected violation is detected, conflict mediation begins. If the TTP determines that a violation has occurred, multiple actions are possible, depending on the given monitoring policies. One possibility is that the offending participant is penalized and the service terminated. Another possibility is that service continues under a renegotiated SLA (see Section 4.1.7). For instance, the consumer receives a reduction in price for a reduction in the Quality of Service (QoS).

4.3.5 Use Case Scenario

This section describes a use case scenario that illustrates self-adaptive SLA monitoring with a provider P, a consumer C and a TTP.

P and C agree upon an SLA for a set of services. As part of this SLA, P and C agree to use self-adaptive monitoring mediated by a chosen TTP. At this point in the lifetime of service provisioning, both participants have little knowledge about one another, so the perceived risk level is high. According to individual policies, both participants begin service monitoring in the active mode, with an interval of 10 seconds.

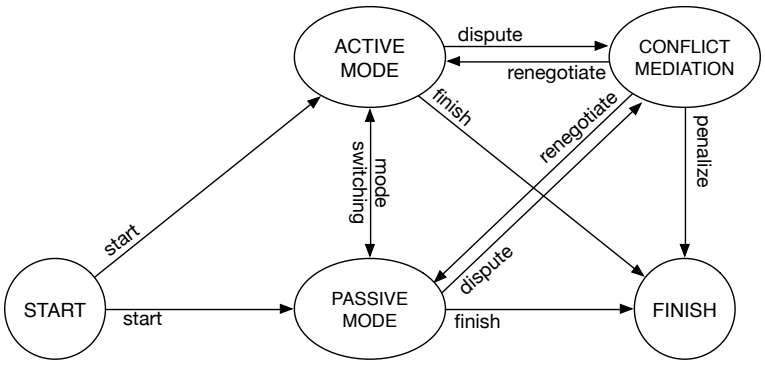

Figure 4.9: State diagram of self-adaptive monitor (adapted from [84]).

After a period of 5 minutes without violation, the monitoring policy automatically decreases the perceived risk level, as trust grows between the participants. This change in risk level is reflected in the monitoring process with a decrease the measurement interval to 20 seconds.

After another period of 5 minutes of successful interaction, the monitoring policy automatically decreases the perceived risk level again. The change in risk is reflected, according to the monitoring policy, in a switch to a passive mode with an interval of 60 seconds.

After 5 more minutes without violation, trust increases, risk decreases and the measurement interval is extended to 90 seconds. However, during the next period of 5 minutes, C detects that an SLA violation has occurred. The TTP is contacted to perform conflict mediation (see Section 4.3.4) as agreed upon in the SLA. The TTP discovers that P has violated the terms of the SLA and P is penalized. Both participants nevertheless decide to reinstate the SLA. The perceived level of risk is affected by the violation, therefore the monitoring policy automatically switches to an active mode of monitoring with an interval of 10 seconds.

Section 4.5 describes an implementation of this use case scenario. Figure 4.16 illustrates the results of experimentation with the same use case scenario.

4.4 Framework Implementation

The self-adaptive monitoring framework is implemented in the AgentScape middleware (see Section 2.3.2). Participants (e.g. consumers and providers) are represented by autonomous, software agents. This middleware is chosen for

implementation and experimentation because it provides a high level programming framework for developing distributed applications. AgentScape offers a programming interface and communication framework that is well suited for rapid prototyping the distributed monitoring framework.

Moreover, the AgentScape architecture fulfills the requirements for a Trusted Computing Base [89], including clearly defined and enforceable security policies, role-based access controls, identification and authentication mechanisms and audit trails [134]. These features make it possible to serve as the TTP. In this implementation, asymmetric cryptography is accomplished using the Rivest-Shamir-Adleman (RSA) [144] algorithm. The signing scheme is based on the Boneh-Lynn-Shacham (BLS) [15] algorithm[1].

4.4.1 Framework deployment

The main monitoring components discussed in Section 4.1.2 are integrated into AgentScape. The monitoring process is contained in an autonomous software agent that performs the role of TTP. Monitoring sensors (TMM) are inserted in each Host Manager, Service and Web Service (WS) Gateway. These strategic locations are chosen for access to relevant monitoring metrics. Host Managers control access to local (hardware) resources, such as CPU and memory. Integrated TMMs report on usage of these resources. Services control access to local (web) services (e.g. currency converter service). An integrated TMM monitors bandwidth, network throughput volume, response time and other relevant QoS metrics. Similarly, a TMM located in a WS Gateway measures metrics relevant to external (web) services regulated by that particular WS Gateway.

Figure 4.10 provides an overview of the monitoring framework in AgentScape. The Location Manager first collects resource availability data from all Host Managers and conveys this to the provider agent (see step 1). The provider agent P representing the given service, in turn, collects these data from the Location Manager. This information is required before services may be offered to the consumer agent C during the SLA negotiation (see step 3). Additionally, this information could affect QoS guarantees. For instance, if a certain service is available but currently in use by several other consumers, the provider may choose to offer the service for a lower price with lower QoS guarantees.

[1]The BLS implementation uses existing code provided by Dalia Khader and the Java Pairing Based Cryptography Library (jPBC) provided by Angelo de Caro (`http://gas.dia.unisa.it/projects/jpbc/`). Source code is available upon request.

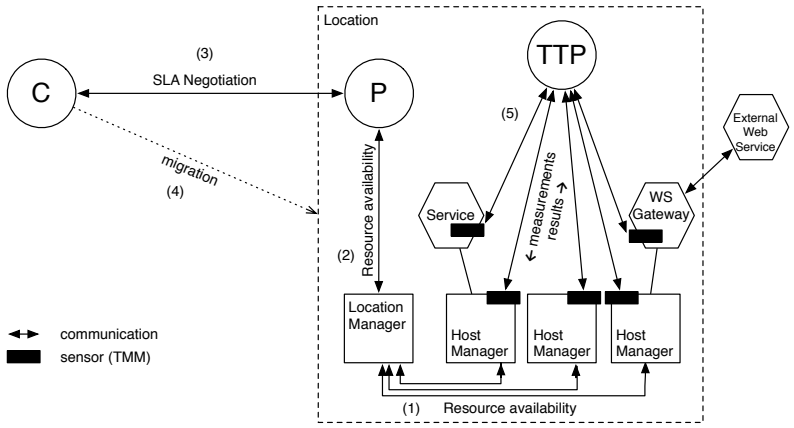

Figure 4.10: Monitoring framework deployment in AgentScape.

After negotiation, the consumer agent optionally migrates to this Location to access the chosen service (see step 4). Depending on the chosen monitoring policy, services are monitored either actively or passively. An autonomous TTP agent performs active monitoring and periodically communicates directly with TMMs relevant to the chosen service. The TTP requests measurements and collects results (see step 5). These results are analyzed and violations are detected. The inner workings of the TTP resemble the Monitoring Process depicted in Figure 4.1. Consumer and provider agents perform passive monitoring directly with one another, with no additional TTP.

To improve scalability, the TTP supports several modes of decentralization. Figure 4.11 shows the different lines of communication for the different modes. In the simplest mode, a single TTP agent is responsible for all Locations in a given World. This single TTP agent (see Location A) communicates with all TMMs in all Locations to monitor services. This mode is highly centralized and better suited to small deployments. An additional mode creates a single TTP agent per Location. All TMMs in a given Location are controlled by this TTP (see Locations A, B and C). This is better suited to medium deployments. The final mode creates multiple TTP agents per Location based on demand (see Location X). If the number of active SLAs increases beyond a given threshold, an additional TTP agent is created. Current SLAs are monitored by the first TTP and new SLAs are monitored by the newly created TTP until both TTPs are equally loaded. Once both TTPs are responsible for the same number of SLAs, new SLA monitoring responsibilities are divided evenly across them following a round-robin strategy. If the number of active SLAs increases beyond the given threshold for both TTPs, an additional TTP is created.

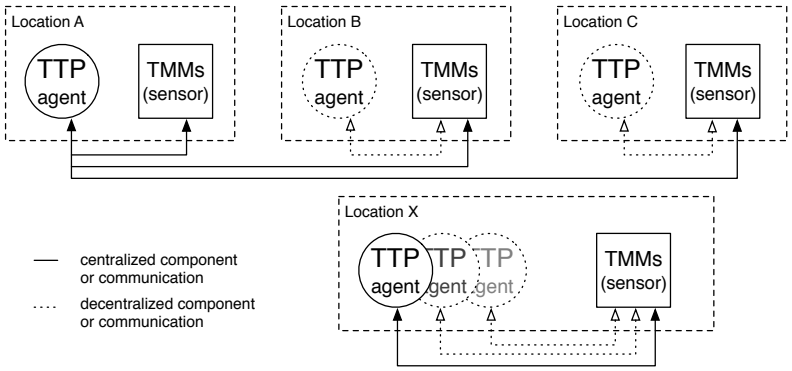

Figure 4.11: Centralized and decentralized TTP modes.

Experiment	Mode	Agents	Interval	Mediation %
Messages_A	Active	2	10, 20, ..., 120	N/A
Messages_P	Passive	2	10, 20, ..., 120	0%, 10%, 30%, 50%
Bytes_A	Active	2	10, 20, ..., 120	N/A
Bytes_P	Passive	2	10, 20, ..., 120	0%, 10%, 30%,50%
ScalingMsgs_A	Active	2-100	30	N/A
ScalingMsgs_P	Passive	2-100	60	0%
ScalingCPU_A	Active	64 - 4096	10	N/A
ScalingCPU_P	Passive	64 - 4096	60	0%
Self-Adaptive	Active & Passive	2	10, 20, 60, 90	N/A

Table 4.2: Overview of experiments.

Monitoring agents (TTP) are robust against failures. These agents are automatically restarted with current data if they crash unintentionally. Data persistence is achieved by routinely writing crucial information, such as the state of the SLA, to the agent container on disk. Active SLAs and monitoring data are recovered from the persistent container file and monitoring continues with minimal disruption.

4.5 Experimental Validation

Several experiments measure communication, CPU overhead and scalability. The first set of experiments measures communication overhead. The second set of experiments examines the scalability of each monitoring mode. The final set illustrates using modes together as a self-adapting monitor, as proposed in this dissertation. Table 4.2 provides an overview of these experiments.

Note that these experiments show the difference in overhead between active and passive modes. However, these experiments do not attempt to define what level of CPU or network overhead is acceptable or unacceptable. This is a subjective threshold that is highly dependent on the context and application. For instance, if resources are cheap and abundant, an agent might find it acceptable if the monitor uses 99% of the CPU. However, in an environment with limited network bandwidth, an agent may find it unacceptable if the monitor uses more than 5% of the bandwidth. An agent customizes his/her monitoring policy to fit a specific context and application.

4.5.1 Communication Overhead Experiments

The communication overhead experiments include Messages_A, Messages_P, Bytes_A and Bytes_P. These experiments compare active and passive modes in terms of the number of messages per minute and the amount of network traffic in bytes per minute. All experiments run for 30 minutes. All monitoring related messages are counted and measured (size in bytes). The total count and size are then divided by 30 to produce the results per minute. Each experiment is repeated with measurement intervals ranging from 10 to 120 seconds, in increments of 10.

In the *Messages_A and Bytes_A* experiments, one consumer and one provider create an agreement that uses active monitoring with a constant, specified interval. The experiment uses one TTP and two measurement sensors.

In the *Messages_P and Bytes_P* experiments, one consumer and one provider create an agreement that uses passive monitoring with a constant, specified interval. The experiment uses one TTP. Both of these experiments are repeated with an additional variable to measure the impact of mediation on communication overhead. Each round of experimentation uses a different mediation percentage (0%, 10%, 30% and 50%). A mediation percentage of 10 indicates that, on average, conflict mediation will be requested 10 percent of the time.

4.5.1.1 Experimental Environment

The communication experiments execute on a single machine. The machine is a SUN SPARC Enterprise T5240 with 2 multicore 1.2GHz CPUs offering 128 hardware threads and 64GB of RAM. This machine runs Solaris 10 and AgentScape middleware. Two AgentScape Locations are created: C and P. A single Consumer agent is hosted at Location C and a single Provider agent is hosted at Location P. A single TTP is hosted at Location P. This set of experiments uses exactly one Consumer and one Producer agent. Active monitoring mode uses two sensors (i.e. Trusted Monitoring Modules).

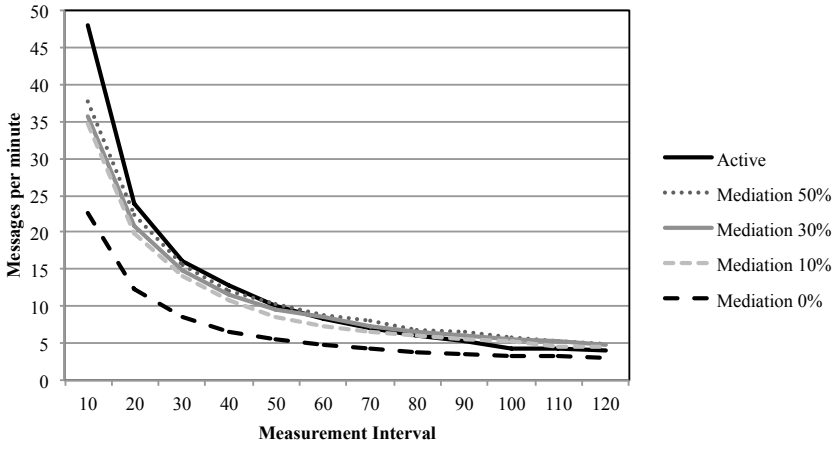

Figure 4.12: Monitoring messages per minute with 2 agents.

4.5.1.2 Experimental Results

The solid, black line in Figure 4.12 depicts the results of the $Messages_A$ experiment. This line indicates the number of messages per minute for active monitoring mode. When the measurement interval is 10 seconds, the monitor uses 48 messages per minute. As the measurement interval increases, the number of messages per minute drops. The number of messages per minute drops below 10 when the measurement interval is greater than or equal to 50 seconds.

The $Messages_P$ experiment results in Figure 4.12 show the number of messages per minute for passive monitoring mode. The baseline (Mediation 0%) shows that when the measurement interval is 10 seconds, monitoring uses 22 messages per minute. The average number of messages per minute drops below 10 when the measurement interval is 30 seconds or more.

Several variations of the experiments have varying percentages of conflict mediation (Mediation 10%, Mediation 30%, Mediation 50%). While there is a significant difference between no mediation at all and 10% or more mediation, there is little difference between the frequency of mediation. When the measurement interval is 10 seconds, the ascending mediation scenarios require an average of 34.6, 35.6 and 37.7 messages per minute, respectively. For all scenarios, this number drops below 10 when the measurement interval is greater than or equal to 50 seconds.

The solid, black line in Figure 4.13 depicts the results of the $Bytes_A$ experiment. This line indicates the network traffic of active monitoring mode in bytes per minute. When the measurement interval is 10 seconds, network

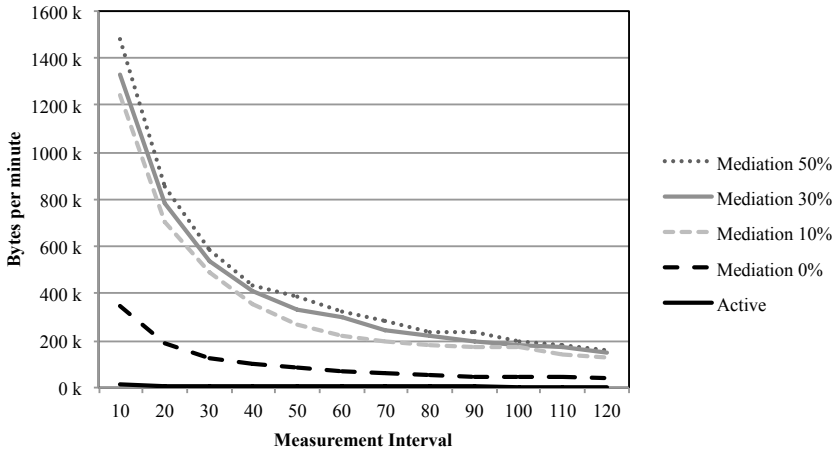

Figure 4.13: Monitoring bytes per minute with 2 agents.

traffic is 16k. Traffic drops below 4k when the measurement interval is greater than 40 seconds.

The $Bytes_P$ experiment results in Figure 4.13 show the network traffic of passive monitoring mode in bytes per minute. The baseline (Mediation 0%) shows the scenario when there is no conflict mediation requested (i.e. participants do not detect violations). When the measurement interval is 10 seconds, monitoring generates 348k bytes per minute of network traffic. Traffic drops below 100k when the measurement interval is 40 seconds or greater.

Several variations of the experiment have varying percentages of conflict mediation (Mediation 10%, Mediation 30%, Mediation 50%). These scenarios generate significantly more bytes per minute on average than the baseline scenario. While there is a significant difference between no mediation at all and 10% or more mediation, there is little difference between the frequencies of mediation. When the measurement interval is 10 seconds, the mediation scenarios generate an average network traffic of 1,243k, 1,328k and 1,479k, respectively. For all scenarios, traffic drops below 400k when the measurement interval is 40 seconds or greater.

Tables 4.3 and 4.4 summarize the results. These tables show the average (AVG) values after the experiments were repeated 10 times. The relative standard deviation (RSD) indicates the variation of values across the 10 repetitions.

Table 4.3: Summary of messages per minute results.

		Messages per minute											
Interval		**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**	**110**	**120**
Active	AVG	48.0	24.0	16.0	12.8	10.1	8.3	6.9	6.1	5.3	4.3	4.3	4.0
	RSD	0.0	0.0	0.0	0.1	0.1	0.1	0.0	0.0	0.0	0.3	0.1	0.0
Passive Med 0%	AVG	22.7	12.3	8.5	6.7	5.6	4.8	4.3	3.7	3.5	3.2	3.2	2.9
	RSD	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Passive Med 10%	AVG	34.6	19.9	14.2	10.8	8.5	7.3	6.5	5.9	5.6	5.4	4.6	4.5
	RSD	0.0	0.0	0.1	0.1	0.0	0.1	0.0	0.1	0.1	0.0	0.1	0.2
Passive Med 30%	AVG	35.6	20.9	14.9	11.6	9.5	8.6	7.3	6.6	5.9	5.6	5.2	4.8
	RSD	0.0	0.1	0.1	0.0	0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1
Passive Med 50%	AVG	37.7	22.3	15.7	12.1	10.4	8.9	8.0	6.9	6.6	5.8	5.4	4.9
	RSD	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.1

4.5.1.3 Discussion of results

The communication overhead experiments indicate a significant difference between active and passive mode with regard number of messages per minute and the total network traffic generated (see Figure 4.12). Passive mode clearly uses fewer messages per minute than active mode. This difference is clarified with an explanation of the inner workings of passive mode. When no conflict mediation is requested, communication occurs directly between consumer and provider. No TTP or TMM are involved. This results in fewer messages per minute than the corresponding active mode.

Figure 4.13 illustrates the difference in the network traffic generated by active and passive mode. Passive mode clearly generates significantly more bytes per minute than active mode, especially when conflict mediation is requested. If no conflict mediation is requested, consumer and provider exchange encrypted portions of individual audit logs. These audit logs are much larger than the simple metrics exchanged in active mode. If conflict mediation is requested, all agents send the entire audit log to the TTP, resulting in a substantial increase in network traffic.

4.5.2 Scalability experiments

The scalability experiments include ScalingMsgs$_A$, ScalingMsgs$_P$, ScalingCPU$_A$ and ScalingCPU$_P$. These experiments investigate the ability of each monitoring mode to scale in distributed environments.

In the *ScalingMsgs$_A$* and *ScalingMsgs$_P$* experiments, consumers and providers create agreements that use either only the active monitoring mode with

Table 4.4: Summary of bytes per minute results.

Bytes per minute							
Interval		**10**	**20**	**30**	**40**	**50**	**60**
Active	AVG	16.6k	8.3k	5.5k	4.4k	3.4k	2.8k
	RSD	0.0	0.0	0.0	0.1	0.1	
Passive Med 0%	AVG	347.8k	185.7k	127.5k	98.4k	81.8k	69.4k
	RSD	0.0	0.0	0.0	0.0	0.0	0.0
Passive Med 10%	AVG	1,243.3k	700.4k	491.9k	357.6k	269.8k	217.5k
	RSD	0.0	0.1	0.2	0.2	0.1	0.3
Passive Med 30%	AVG	1,327.8k	782.8k	535.9k	407.1k	328.0k	296.8k
	RSD	0.0	0.1	0.1	0.0	0.1	0.1
Passive Med 50%	AVG	1,479.0k	856.6k	585.0k	433.1k	383.4k	321.8k
	RSD	0.0	0.1	0.1	0.1	0.1	0.1

Bytes per minute (continued)							
Interval		**70**	**80**	**90**	**100**	**110**	**120**
Active	AVG	2.4k	2.1k	1.8k	1.4k	1.4k	1.3k
	RSD	0.0	0.0	0.0	0.2	0.1	0.0
Passive Med 0%	AVG	61.0k	52.7k	48.6k	44.4k	44.4k	40.3k
	RSD	0.0	0.0	0.0	0.0	0.0	0.0
Passive Med 10%	AVG	192.2k	181.5k	173.8k	169.9k	138.2k	127.8k
	RSD	0.1	0.3	0.3	0.1	0.1	0.3
Passive Med 30%	AVG	243.5k	220.0k	193.9k	183.1k	171.4k	148.0k
	RSD	0.2	0.2	0.3	0.3	0.2	0.2
Passive Med 50%	AVG	285.3k	237.3k	234.7k	194.4k	180.7k	158.5k
	RSD	0.1	0.1	0.1	0.2	0.1	0.1

a fixed interval of 30 seconds or only the passive monitoring mode with a fixed interval of 10 seconds. These values were chosen based on the performance of previous experiments. These experiments use one TTP and two TMMs. As with the previous set of experiments, each of these experiments is run for 30 minutes and results are averaged. Each experiment is repeated with the number of agents ranging from 2 to 100, in increments of 2.

In the $ScalingCPU_A$ and $ScalingCPU_P$ experiments, a consumer agent and a provider agent are launched simultaneously on 32 separate Locations on 32 separate nodes. A consumer chooses a provider from amongst the Locations using a random function with a uniform distribution. The chosen provider is

always located on a different node than the consumer. After choosing a provider, the consumer creates an agreement that uses monitoring. ScalingCPU$_A$ uses active monitoring mode with an interval of 10 seconds. ScalingCPU$_P$ uses passive mode with an interval of 60 seconds. These values are chosen based on the performance of previous experiments.

A new consumer agent and provider agent are launched every 5 seconds until the desired number is reached. For the first run, this number is 64 (2 per node). The number of agents increases by 64 on each consecutive run until reaching 4096 (128 per node). The entire process is repeated 5 times and the CPU load results for all 32 participating nodes are averaged.

4.5.2.1 Experimental environment

The first set of scalability experiments (ScalingMsgs$_A$, ScalingMsgs$_P$) use the same machine and configuration as the communication overhead experiments.

The second set of scalability experiments (ScalingCPU$_A$, ScalingCPU$_P$) use the Distributed ASCI Supercomputer [8] - version 4 (DAS-4)[2]. A single AgentScape World combines 33 independent nodes of the DAS-4 grid. Each node has a 2.4 GHz processor, 24GB of memory and runs CentOS Linux. Gigabit ethernet connects the machines. One node runs only a single AgentScape Lookup Service that provides listings of all known Locations. On the remaining 32 nodes, a single AgentScape Location runs with a single TTP Agent. Thus, there are 32 TTP Agents in the World.

4.5.2.2 Experimental results

Figure 4.14 shows the *ScalingMsgs$_A$* and *ScalingMsgs$_P$* experiment results. These results indicate the messages per minute of the two monitoring modes as the system scales to 100 agents. Both modes scale linearly, but passive mode requires fewer messages, overall.

Figure 4.15 presents the results of the *ScalingCPU$_A$* and *ScalingCPU$_P$* experiment results. These results demonstrate the scalability of both modes of monitoring in a highly distributed environment. Active mode performs better than passive mode. Even with 4096 agents in the system, each being monitored at 10 second intervals, the average CPU load remains below 0.5%. The passive mode approaches 4% with 4096 agents being monitored at 60 second intervals.

[2] `http://www.cs.vu.nl/das4/`

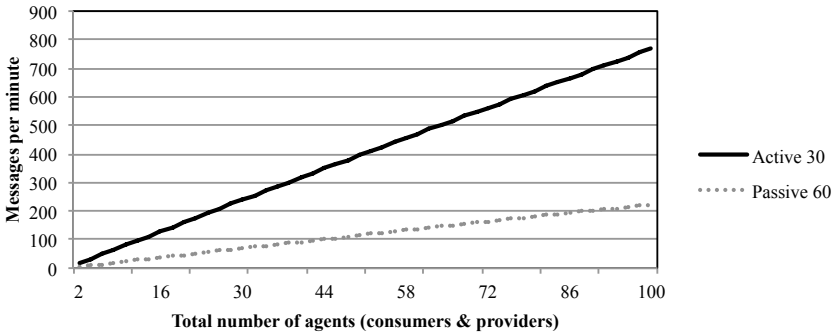

Figure 4.14: Message overhead with increased scalability

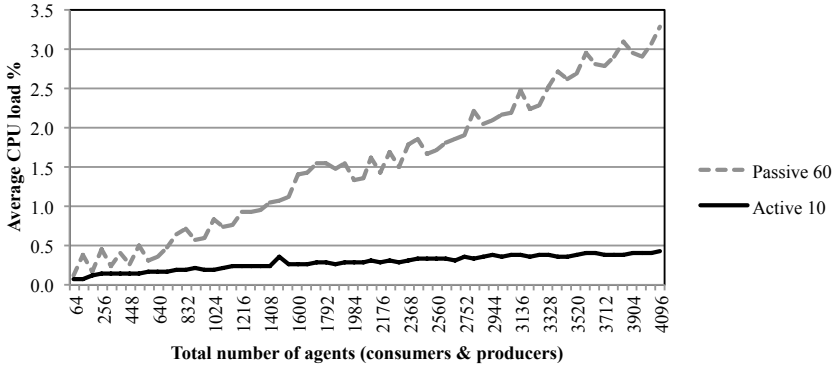

Figure 4.15: Average CPU load of large scale monitoring on DAS-4.

4.5.2.3 Discussion of results

Figure 4.14 indicates that both monitoring modes scale linearly, in terms of communication overhead. Doubling the number of agents doubles the number of messages. Figure 4.15 indicates that both monitoring modes scale in large, distributed systems with only minimal computational overhead. In active mode, CPU load is affected by the interval of measurement. In passive mode, CPU load is determined by both the interval of measurement and the frequency with which mediation is requested.

The passive mode has a higher average CPU load than the active mode. The CPU-intensive cryptographic computations account for this difference[3]. Active mode requires no cryptography, as security is guaranteed by the TTP.

[3]In particular, the BLS implementation is chosen only for its functionality. This code is not optimized for production level systems.

What is not visible in these particular figures is the level of interaction with the TTP. In passive mode, there is only a minimal amount of interaction with the TTP (e.g. exchanging cryptographic keys). The consumer and provider perform all computations, including measurements and cryptography. In contrast, active mode relies on the TTP to perform all measurements. With this in mind, the active mode (see Figure 4.15) essentially depicts the CPU load at the TTP. The corresponding CPU load at the consumer and provider is insignificant. In contrast, the passive mode essentially depicts the average CPU load at the consumer and provider. The corresponding CPU load at the TTP is insignificant.

4.5.3 Self-adaption experiment

The *Self-Adaptive* experiment combines both modes and provides an example of the self-adaptive monitoring approach proposed in this dissertation. This experiment shows the changes overhead as the monitoring process dynamically switches between intervals and modes.

One consumer and one provider create an agreement that begins with an active monitoring mode with an interval of 10 seconds. Both agents use the same monitoring policy that specifies that after 5 minutes without violations, the Risk Level decreases. This decrease is reflected by increasing the monitoring interval or changing modes. When an agent detects an SLA violation, mediation is requested. The result of mediation is to increase the Risk Level. This increase is reflected in the monitoring level by resetting to the initial configuration of active mode with an interval of 10 seconds. The self-adaptive experiment is repeated 10 times and the CPU load results are averaged.

4.5.3.1 Experimental environment

The Self-Adaptive experiment runs on a network of two machines connected across gigabit ethernet. The first machine has a 2.0GHz dual core CPU and 1GB of RAM. The second machine has a 2.0GHz dual core CPU and 2GB of RAM. Both machines run Ubuntu Linux and AgentScape middleware. Two AgentScape Locations are created, one on each machine: C and P. A consumer agent is hosted at Location C and a Producer agent is hosted at Location P. A single TTP is hosted at Location P.

4.5.3.2 Experimental results

The Self-Adaptive experiment results in Figure 4.16, provide an overview of the self-adaptive monitoring process. Average CPU usage is indicated by the solid, black line. Average messages per minute is indicated with shaded, grey

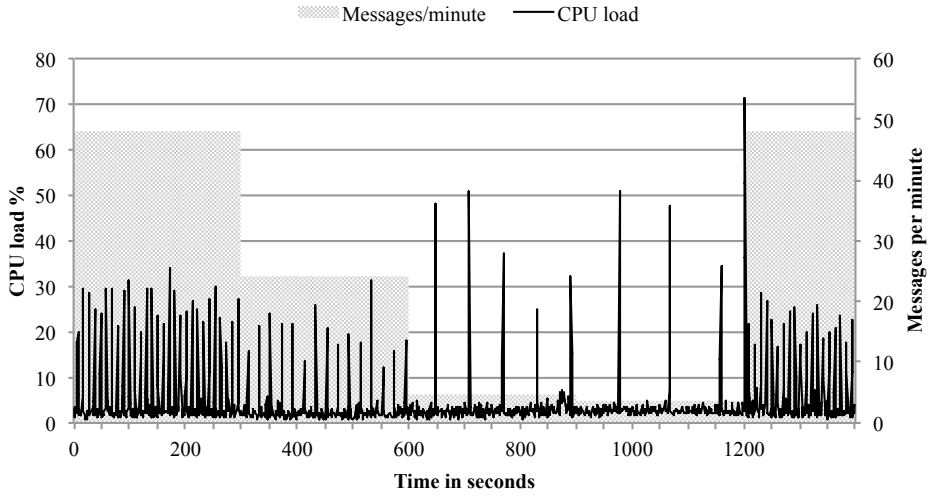

Figure 4.16: Overhead of self-adaptive monitoring with two agents.

bars. In active modes, CPU usage has a consistent pattern reflecting the 10 and 20 second monitoring interval, respectively. The number of messages per minute is relatively high. In passive modes, consistent CPU usage reflects the 60 and 90 second monitoring interval, respectively. The number of messages per minute is substantially lower. CPU usage surges when conflict mediation is requested. This surge reflects the additional cryptography required to verify the messages and aggregate signatures in the audit logs.

4.5.3.3 Discussion of results

This experiment demonstrates how monitoring overhead is reduced when agents feel there is little threat of violation. This experiment uses a self-adaptive monitoring policy that starts with a high level of perceived risk. After a period of time without incident, the level of perceived risk lowers. This is reflected by increasing the measurement interval and thus reducing overhead. Eventually, this leads to a switch to passive mode, which reduces dependence on a TTP.

If a violation is detected, mediation is requested. This may result in a higher level of perceived risk and a correspondingly high level of monitoring. This significantly increases the message and CPU overhead but offers higher assurance. By adjusting the monitor to match the level of perceived risk, overhead is substantially reduced when possible.

4.6 Related Work

There is growing interest in the monitoring of SLAs in distributed environments. The Lattice Monitoring Framework (LMF) detailed in [35] has been designed as a resource monitor for virtualized environments, such as the Cloud. This framework makes a clear separation of roles between service providers and infrastructure providers. For instance, Dropbox[4] is a service provider and Amazon S3[5] is the infrastructure provider. LMF aims to monitor services across multiple infrastructure providers (e.g. across different administrative domains). However, it is not clear which participant has access to the actual measurements, and therefore, could potentially tamper with measurement results. In contrast to the approach presented in this dissertation, LMF is designed specifically as a tool for the provider and is thus hidden from the consumer. No Trusted Third Party (TTP) is employed to ensure that monitoring results are correct. Thus, the consumer must trust that the provider performs measurements correctly and does not tamper with the results.

The SLA@SOI project[6] incorporates the monitoring framework presented in [37]. This approach also focuses on the provider's perspective. The framework collects and stores historical monitoring data during negotiation to evaluate the SLA offers made by a customer. Furthermore, the system checks the SLA at runtime to determine if the terms are capable of being monitored at all. This is referred to as the *monitorability* of the SLA. Each service provider has a list of terms for which the required measurement logic and sensors are available. If a new SLA has terms that are not compatible with this list and therefore not *monitorable* by this particular provider, the SLA is rejected. The system is completely controlled by the provider and is opaque to the consumer. The system has no objective party (e.g. TTP), thus consumers must implicitly trust that the provider does not manipulate monitoring results.

[54] proposes an SLA framework using SLA as a mechanism for service providers to keep track of QoS commitments. Providers monitor system metrics to determine if QoS commitments are being met and then dynamically provision resources to prevent over- or under provisioning. [160] describes a similar framework. The service provider monitors SLAs internally for possible violations. If necessary, the system automatically replicates services to meet obligations, such as response time and transaction rate. In effect, both of these systems are SLA driven load-balancing systems. They provide tools for the provider to optimize system load while meeting SLA obligations. In

[4] Dropbox is a file hosting service that, in turn, uses the Amazon S3 storage service: `http://www.dropbox.com/`

[5] Amazon S3 is an online file storage service: `http://aws.amazon.com/s3/`

[6] `http://sla-at-soi.eu/`

contrast to the approach presented in this dissertation, there is no TTP and the systems are only designed for internal QoS monitoring at the provider. Fairness towards the consumer and transparency of the monitoring process are not considered.

A mechanism for enforcing SLAs for scientific computing is presented in [115]. This mechanism is tailored to deadline-driven batch jobs, such as in scientific compute grids. At runtime, a fuzzy prediction algorithm estimates the amount of resources needed for each job. Using this estimation, the system determines if an SLA can be met and accepts or rejects a job accordingly. If a job is accepted, it is assigned a software agent. This agent monitors a job during its lifetime and dynamically increases resources to ensure that its SLA is not violated. Similar to some of the frameworks described above, this system is not designed to ensure transparency of the monitoring process for consumers. Consumers must implicitly trust the monitoring results from the provider.

The self-adaptive SLA monitoring framework proposed in this dissertation differs from the frameworks described above. First, the approach proposed in this dissertation explicitly establishes the role of a dedicated TTP. This ensures that monitoring results are accurate and neither agent is able to manipulate the outcome.

Using a TTP to guarantee the objectivity of monitoring results also affects the relationship between consumer and provider. The monitoring framework views the consumer and the provider as equal participants. As such, the monitoring framework is equally transparent to both participants. This allows both participants, not just the provider, to trust the monitoring results. Both participants have equal access and equal influence on the monitoring data. This equality balances the power between participants. In the frameworks described above, the provider holds a more powerful position as sole controller of monitoring data. The consumer is placed in a weaker position and must assume the provider is honest and does not secretly manipulate monitoring data.

In contrast to the monitoring approaches discussed thus far, an approach that acknowledges and addresses the issue of trust is the QoS-MONaaS framework presented by [145]. This is an extension to the SRT-15 middleware[7]. This system is specifically designed to support trustworthiness of monitoring results. This system uses an anonymizer function to ensure fairness in measurement. Essentially, the anonymizer strips identifying information from measurement requests. The claim is that because the monitoring framework does not know the identity of an agent that requests measurement results,

[7] http://www.srt-15.eu

there is no incentive to tamper with these results. For example, in a scenario where both a consumer and a provider request monitoring results, the monitoring framework is unable to intentionally provide correct results to the provider and tampered results to the consumer. The framework would have no way of knowing which agent would receive which set of results. While this monitoring approach addresses the issue of trust, it does not address the issue of adaptation.

In contrast to the monitoring frameworks described above, some monitoring techniques are able to dynamically adapt monitoring policy at run-time based on environmental limits or changes in priorities. For instance the monitoring approach proposed in [83] collects system notifications from distributed nodes and dynamically adjusts the frequency of notifications, based on CPU load. The higher the load (e.g. more agents in the system), the lower the frequency of notifications. [110] describes another example of dynamic monitoring in the form of an adaptive system monitor. This monitoring process attempts to reduce monitoring overhead by pre-selecting and focusing on key metrics. Only when an anomaly is detected in one of these key metrics, does the monitoring process adapt by increasing the number of related metrics that are continuously monitored. Effectively, this monitoring process is able to 'zoom in and out' of areas when problems are detected.

The self-adaptive SLA monitoring framework proposed in this dissertation differs from these two approaches. First, not only does the approach proposed in this dissertation react to changes in system overhead, but also to changes in the level of perceived risk. Secondly, rather than only adjusting the measurement interval, the monitoring process is able to switch between two major modes of monitoring: active and passive. In the passive mode, dependence on the TTP is significantly reduced. This dependency reduction results in a reduction in costs and an increase in the ability of the system to scale.

4.7 Conclusions

Services are monitored to ensure that SLA obligations are being met by all participants. Monitoring offers assurance that SLA violations are detected and the responsible participant is identified. This makes it possible to penalize the offending participant or take corrective action.

Service monitoring in open environments presents several challenges, including trust, scalability and dynamism. Participants are not implicitly trustworthy and may wish to deceive other participants. To prevent a participant from influencing monitoring results this dissertation proposes the use of an external, impartial monitoring service. An example of such a service

is a Trusted Third Party (TTP). A TTP uses Trusted Monitoring Modules (TMM) to perform measurements securely such that results are trustworthy to all participants.

Open environments are inherently distributed across many geographical areas and administrative domains. In such environments, centralized monitoring approaches is a possible bottleneck. Therefore, this dissertation proposes distributing monitoring activities and data across multiple participants. This is accomplished by the use an alternative monitoring technique, referred to as passive monitoring. Passive monitoring distributes monitoring responsibilities across the consumer and provider. This reduces interactions with the TTP and thus allows the TTP to handle more clients concurrently. The issue of trust is addressed using several cryptographic primitives.

Another challenge of open environments is that of dynamism. Open systems are subject to contestant change, including participant reputation and the relative importance of a particular service transaction. These changes may affect a participant's perceived level of risk. This is a function of the likelihood of failure and the impact of such a failure. This directly affects a participant's monitoring needs. This dissertation proposes that a monitor should adapt itself to those monitoring needs.

A self-adaptive monitor proposed in this dissertation dynamically switches between the traditional active mode and the cryptographic passive mode. Further adaptation is possible by fine-tuning the measurement interval. The monitor increases the level of monitoring when participant's perceived level of risk increases, and vice versa. This results in high monitoring assurance when needed and reduced monitoring overhead when possible. The self-adaptive monitor is implemented and experimentally validated in the AgentScape middleware.

CHAPTER 5

Use Cases: Smart Energy Grid & Cloud Computing

The automated negotiation and monitoring techniques presented in the previous two chapters form an important part of an integrated framework. This framework supports marketplace agents throughout the lifecycle of SLAs, including service discovery, agreement negotiation, provisioning and monitoring. The framework provides a structured and trusted platform for agents to provide or consume services in dynamic, open environments.

The applicability of the negotiation and monitoring framework is demonstrated in this chapter for two use case scenarios: (1) the Smart Energy Grid and (2) Cloud computing. These two scenarios are chosen as they represent socio-technical systems [181], involving both technical aspects (e.g. software, hardware) and social aspects (e.g. usability, user participation and acceptance). These scenarios contain technical challenges, including highly dynamic resources and open, untrusted networks. In addition, these scenarios must accommodate human users, enable human participation and encourage human acceptance. Achieving full potential of socio-technical systems requires addressing both social and technical challenges.

In the case of the Smart Energy Grid, climate and pollution considerations drive adoption of renewable resources. (Semi) autonomous, automated

This chapter is based on three published papers [30–32].

technologies, such as proposed in this dissertation, enable more efficient use of renewable resources and reduce wasted energy. In the case of Cloud computing, both companies and private citizens increasingly rely on Cloud-based solutions. (Semi) autonomous, automated technologies reduce costs and increase efficiency.

This chapter proceeds as follows. First, an overview introduces the main concepts of future energy markets, including challenges and solutions. Secondly, this chapter presents an multiagent solution to automate an energy market. Thirdly, a use case scenario demonstrates the applicability of the techniques proposed by this dissertation to energy markets. Fourthly, an overview introduces the main concepts of Cloud computing, including the challenges of standardization. Fifthly, this chapter presents the Intelligent Cloud Resource Allocation Service (ICRAS) that combines the negotiation and monitoring techniques proposed by this dissertation. Sixthly, a use case scenario demonstrates the applicability of ICRAS to the Cloud services market.

5.1 Dynamic Services in the Smart Energy Grid

Global investment in renewable energy has grown by more than 600% since 2004 [104]. Growing concerns for the environmental impact of fossil fuels (e.g. CO_2 emissions) fuel this trend. Many countries are investing in renewable (or 'green') energy sources and new, advanced infrastructure. The European Union has set a goal for its member countries to use renewable sources for at least 20% of total energy consumption by 2020 [36]. In Germany, the government recently unveiled plans to invest 20 billion euro in a new energy network to support a goal of 80% energy from renewable resources [98]. In the Netherlands, the percentage of total energy production from renewable sources has quadrupled in the past decade[1]. The energy landscape of the future is becoming 'greener', but the influx of renewable energy presents a challenge to the current energy grid. Automated, coordination mechanisms and incentives meet these challenges, by reducing wasted overcapacity for producers, lowering prices for consumers and increasing overall utilization of green energy sources. The following sections describe the characteristics and technologies of this green energy future.

5

[1] Centraal Bureau voor de Statistiek. `http://www.cbs.nl/nl-NL/menu/themas/industrie-energie/publicaties/artikelen/archief/2010/2010-3105-wm.htm`. Accessed: September 2012

5.1.1 Future Energy Markets

As more green energy sources are harnessed, it becomes more difficult for future energy markets to utilize them efficiently. The energy market attempts to match energy production and consumption (i.e. supply and demand). The balance of energy generation and production is a substantial challenge even when using traditional, continuous energy sources, such as coal power. Green energy sources, such as wind power, further complicate this balance as they are intermittent and unpredictable.

Intermittent production complicates power generation and transmission planning. This, in turn, complicates the process of matching supply and demand, leading to market inefficiency. For instance, energy demand at the office rarely coincides with weather conditions at the wind park. It is estimated that most wind farms have an effective capacity of 10% of full potential [129]. This means that only 10% of full potential power reaches consumers, when consumers need it, due to planning and coordination difficulties. Such mismatches between supply and demand result in an inefficient market with substantial wasted potential energy.

The challenge of intermittent generation is exacerbated by a lack of inexpensive, abundant energy storage technologies. Traditional chemical batteries are too small and too expensive to handle the current storage needs. Demand is increasing for wider application of alternative technologies, including mechanical flywheels [91], new battery construction (e.g. liquid metal batteries[2]), pumped hydroelectric energy storage (PHES)[3] and the use of electric vehicles for energy storage [128]. However, at present no cost-effective techniques exist to store electrical energy on a large-scale. Therefore, energy is either consumed at virtually the same moment it is generated or it is wasted.

Distribution of energy generation will also change with the influx of green technologies. Rather than the current paradigm of centralized power stations, a larger percentage of power will be generated by distributed resources. For instance, consumers install solar panels and micro combined heat and power (micro CHP) stations[4]. Currently, overcapacity generated by these distributed

[2]Clean Technica. `http://cleantechnica.com/2012/02/20/mit-liquid-batteries-for-utilities-could-make-renewables-competitive-and-it-is-not-lithium-ion/`. Accessed: June 2012.

[3]PHES stores energy using electric powered pumps to push water uphill when electricity (e.g. from a wind farm) is plentiful and demand is low. When demand is high and electricity is scarce, water is released to flow downhill, through turbines to generate electricity [38].

[4]Micro CHP is a small gas powered turbine installed in a residential area, near the end customers, that converts gas into both electricity and heat. Performing the conversion near the end users reduces transmission losses and allows efficient use of heat produced as a by-product of conversion [45].

resources is sold back to a single retailer. This retailer is then confronted with the complex task of reselling and redistributing this sporadically generated energy. To reduce the complexity on centralized management (i.e. retailer), groups of small-scale producers and consumers form microgrids or Virtual Power Plants (VPPs) that supply each others' energy demands, independently from the rest of the energy grid [66, 138].

To cope with the added complexity of intermittent energy sources and distribution of production, a new energy management approach has been proposed: the Smart Grid [61, 138]. Two key attributes of the Smart Grid are (1) bi-directional flow of information and (2) demand-side management. Information flow includes real-time metering of user consumption (e.g. smart meters[5]) such that producers quickly detect and respond to changes in demand. Information flow also includes providing consumers with real-time data about energy availability (e.g. current production levels at the local wind farm) and price (e.g. demand is low, thus the current price is reduced). Information flow is critical to enabling consumer participation.

The following sections introduce two techniques for improving efficiency of future energy markets, increasing green utilization and reducing wasted energy.

5.1.1.1 Demand Side Management

One possible solution to market inefficiency, caused by intermittent generation, is Demand Side Management (DSM). DSM is a technique of dealing with fluctuations in energy production by modifying demand to match supply [62, 124, 161]. As production of energy cannot be shifted in some cases (e.g. wind power), DSM attempts instead to shift demand at the consumer end. Demand shifting occurs when an energy consumer reacts to changes in the energy market (e.g. price, availability and so on) by increasing or decreasing consumption.

Traditionally, the complex task of matching supply to demand is handled entirely by the producers of energy. DSM enables end consumers to participate in this task by temporally shifting energy consumption in response to signals from the market, such as price. Typically, economical incentives encourage consumers to reduce consumption during peak periods, when energy is scarce and expensive. One form of DSM is *time-of-use schemes* that offer different energy prices based on time of day. Typically, nighttime prices are substantially lower than daytime prices due to the demand of businesses during working hours. A consumer is thus economically rewarded for running

[5] A smart meter is a device to measure electricity usage and communicate usage information to the consumer and/or energy provider in (semi) real-time [170].

washing- and drying machines during the night rather than during the day. Unfortunately, time-of-use schemes have unwanted side-effects that result in significant usage peaks as soon as the lower tariff period begins [138]. A more dynamic approach is required.

DSM offers many benefits, including the reduction of overcapacity. Currently, energy producers generate approximately 20% more energy than is required at a given moment [161]. This overcapacity is needed to handle unforeseen peaks in demand or problems with production. DSM encourages consumers to avoid creating such peaks and thus the margin of overcapacity is reduced. In some cases, this reduces wasted energy and costs [138].

Another benefit of DSM is the increased utilization of intermittent sources. Renewable sources of energy, such as wind power, produce power intermittently, as opposed to the consistent power provided by traditional sources, such as coal. Intermittent production means that energy is not produced to match consumption patterns, but rather is produced sporadically (e.g. depending on the weather). Due to the unpredictable production schedule of intermittent sources, large amounts of potential energy are wasted [129].

DSM requires communication, incentive and action mechanisms. First, the market must communicate with consumers to inform them of changes in price or (green) availability. Secondly, consumers require an incentive to react to these market signals. One possible incentive mechanism is Real-Time Pricing (RTP) which informs consumers of price changes, in real-time, that reflect current energy availability.

5.1.1.2 Real-Time Pricing

Most often, consumers sign long-term contracts with energy providers for a fixed price per kilowatt hour (kWh). This approach stabilizes prices for consumers, but lacks the incentives necessary for consumer initiated demand shifting. An alternative to this approach is real-time pricing (RTP). Under this scheme, consumers pay the current price of energy as determined by conventional market forces (e.g. supply and demand). When demand is high, the price of energy increases and vice versa. RTP thus offers a tangible incentive to reduce consumption during peak demand periods and shift this consumption to low demand periods. The intended results of RTP are (1) increase market efficiency (i.e. matching supply and demand) (2) reduce overall energy production and (3) empower consumers to take an active role in reducing energy costs.

In some cases, research shows that real-time pricing is effective at increasing market efficiency [16, 40]. Simulations show that real-time pricing is more than five times as efficient as time-of-use schemes [17]. If consumers are aware

of the current price of electricity (e.g. hourly feedback), they reduce consumption when electricity is expensive and shift usage to periods when prices are cheap [3]. If these prices reflect the current market conditions, such as current production capacity and consumer demand, then the result is that consumers reduce demand when supply is limited and increase when supply is abundant. Matching supply to demand reduces wasted energy (i.e. produced but unconsumed) and increases overall market efficiency.

Several energy providers offered RTP to customers on a voluntary basis with varying results [10]. In general, the goals of reducing price and overall production load were met. On average, RTP schemes reduced energy load between 12% and 33%. However, most customers failed to respond to hourly changes in price, unless the price rose above a certain threshold. In most cases, the shortcomings of the RTP scheme are traced to implementation faults rather than a fundamental problem with the theory. For instance, many consumers lacked user-friendly feedback mechanisms (e.g. smart meter) or felt that the correlation between prices and usage was not transparent and therefore lacked trust in the scheme.

An often cited failure of RTP was the California energy crisis of 2000 [16]. This application of RTP led to extreme price volatility with disastrous consequences. Researchers conclude that crisis was the result of a poor market design that allowed producers to exercise market power by artificially reducing supply to increase prices [16].

When applied efficiently, RTP (1) stabilizes demand which simplifies the task of production management and (2) increases market efficiency, thus allowing waste reduction (e.g. supply overcapacity) [17]. In this case, RTP is five times as efficient as time-of-use schemes (e.g. day- and nighttime tariffs). Price volatility is mitigated using a combination of long-term contracts between wholesalers and RTP for end users [16] or a two-part rate scheme that combines an energy quota at a fixed price with RTP for deviations from that quota [10].

Automation technologies currently exist to coordinate information flows and assist consumers with DSM. Modern smart meters combined with intelligent agent technology automates the tasks of (1) monitoring energy prices, (2) negotiating (e.g. double auctions) and (3) coordinating demand shifting [46]. This alleviates the burden of manually reacting to RTP to efficiently reduce costs and better utilize green energy. The following section discusses these automation techniques.

5.1.2 Energy Market Automation

Online markets exist for many areas of commerce, including Web- [26], Grid- [23], Cloud Services [24] and (industrial) Energy Auctions. An online marketplace is essentially a location where providers electronically advertise services and consumers discover and access those services. A marketplace offers additional services and structure, such as an explicit ontology, terminology, protocols and facilitation. Facilitation may include assistance discovering services, negotiating prices or resolving conflicts.

Online markets are organized such that consumers are able to compare services and choose between competing providers. A standardized language is defines (compositional) services. Standardized protocols make it possible to switch seamlessly between providers. A marketplace typically offers some type of directory service, where providers publish available services. Optionally, third-party brokers actively match customers to appropriate providers. Agreements reached between participants are formalized using Service Level Agreements (SLA).

Many market processes are (partially) automated with technologies, such as software agents. Agents are proposed to model and manage complex, distributed systems, such as the energy market [19,51,71,86,173]. Separate agents represent each unique role of the energy market, including energy consumer, producer, mediator, broker and so on. In some cases, agents take on dual-roles. For instance, a residential consumer becomes a producer if solar panels are installed on his/her roof. Additional agents are represent transmission companies, distribution companies and independent system operators.

An important prerequisite for user acceptance is that users trust the Multi Agent System (MAS), especially with regard to critical tasks [72, 73, 101]. Transparency enhancements, auditing mechanisms and third-party certification, are built into the system to promote trust and acceptance.

Agent technology is applied to the energy market to assist human actors with monitoring and responding to real-time information quickly and efficiently. Removing the need for constant human interaction makes it possible to increase the speed and frequency of market interactions. In addition, intelligent automation is often able to react faster than humans in complex, dynamic systems that may be difficult for humans to understand and follow. Another crucial market process is negotiation. After a consumer discovers a provider with a particular service (e.g. wind energy), the two participants attempt reach an agreement regarding the terms and conditions of the service, including price and Quality of Service (QoS) (e.g. uptime, time to repair an outage, minimum green percentage).

5.1.2.1 Multiagent Energy Market

This dissertation assumes that agents represent consumers and providers in future energy markets. A residential consumer is represented by an agent. This agent requires (access to) a consumer's personal preferences, such as *favor local producers* or priority lists such as *solar, wind, biomass, nuclear*. Internal decision rules allow this agent to access a given situation (e.g. evaluate an offer) and take action (e.g. reject the offer). An important instance of decision rules is *negotiation strategy*. This strategy guides the negotiation process and determines when to accept, reject or counter an offer. The agent also requires (access to) information regarding the consumer's historical energy usage. This information influences negotiation strategy.

In addition, this agent has (limited) control of energy consumption in the home. For instance, access to 'smart' appliances such as refrigerators or clothes dryers. At a minimum, the agent must be able to monitor energy usage, turn on and turn off the appliance via some type of network. Additional intelligence indicates priorities and special requirements of individual appliances. For instance, a refrigerator may safely be turned off for a short period of time without serious consequences. However, it may be unacceptable if the television were to turn off in the middle of a show.

A consumer agent finds the best deal among energy providers, given a set of preferences. An example of preferences could be (1) minimize price and (2) maximize green energy. The consumer agent surveys the marketplace to find providers offering suitable services. If one or more suitable providers are found, the agent negotiates the terms and conditions of service with the selected provider(s). If an agreement is reached an SLA is created. The provider then begins service provisioning. During the lifetime of the agreement, the agent monitors the service to ensure that the terms and conditions are met. The monitoring process builds a secure audit log of all transactions. If a violation is detected, penalties are applied. In the case of a dispute, the audit log is consulted to resolve the conflict and advise appropriate action. The entire process is repeated regularly (e.g. every hour) to ensure a consumer has the best price and service.

A consumer agent acts as a smart energy gateway for each end user (e.g. home or office building). This agent interacts directly with agents that represent producers of energy. For instance, one agent represents a company controlling a coal power plant and another agent represents a wind farm. These agents have access to all internal producer data, including current production capacity, profit margins and current demand. The producer agents negotiate sales to consumers based on this local data and negotiation strategy (e.g. maximize usage, maximize profit).

Energy market automation is essential to respond effectively to changes in the market, such as lower prices or abundant green energy, as these changes occur often and without accurate prediction. Automation supports effective DSM in real-time. For instance, based on market conditions, an agent may choose to postpone energy consumption (e.g. clothes dryer). Thus, green energy production better finds green energy demand.

5.1.2.2 Micro-Agreements

To facilitate RTP and DSM, this dissertation proposes micro-agreements (or micro-SLAs). Traditional SLAs (see Section 3.1.2) in the energy domain are negotiated for periods of time ranging from 1 month to several years. For example, most energy providers offer a minimum period of 1 year and incentives (e.g. price reduction per unit) to extend the period to several years. Long-term agreements have the benefit of reducing uncertainty in respective marketplaces [17]. Consumers are offered a fixed price for a given length of time, thus reducing exposure to drastic fluctuations in actual supply and demand. Producers are offered a guaranteed revenue stream that justifies investment in new infrastructure.

Despite the benefits, long-term agreements also prevent participants from reacting in real time to market signals. For instance, increased availability lowers the market price of certain resources offered from a competing provider. However, a consumer with an active, long-term agreement is unable to break the agreement to switch to another provider to take advantage of lower costs. Market signals include price, but also availability of certain, desired resources. For example, an increase in wind energy production lowers the price of 'green' energy from a competing provider. A consumer may prefer green energy, but is unable to break a long-term contract to make use of this resource. In some cases, this results in wasted energy [17, 138].

In contrast to these long-term SLAs, micro-agreements are negotiated for significantly shorter periods, such as 1 hour or 1 day. The shorter period enables consumers to react in real time to market signals. Each period, a consumer (re)evaluates its energy needs, surveys the energy prices from competing providers and negotiates a new micro-SLA for the next period. When this agreement expires, the process is repeated. The consumer selects preferred resources from a provider and takes advantage of lower prices or certain, desired resources (e.g. green energy). For instance, when weather conditions provide for high availability of green energy, consumers are able to quickly migrate energy demand to providers offering this preferred resource. In addition, if a consumer considers all current energy offers to be too expensive, the consumer may consider shifting his/her energy demand.

Micro-SLAs are a useful mechanism that allows both consumers and providers to quickly react to changes in dynamic, open environments. Consumers actively favor green energy resources over other resources and may utilize as much green power as possible by maximizing demand during green peak periods. A secondary effect is the financial rewarding of additional investment by producers in green energy resources. For instance, micro agreements increase effective capacity at a wind farm by enabling consumers to immediately react to increased wind energy production by purchasing energy directly from that wind farm at the time of production. Higher utilization creates higher revenue.

Micro-SLAs also increase resilience in markets by reducing logical distance between consumers and providers. This approach brings (small-scale) stakeholders directly in contact to meet each other's needs. This reduces logical fragmentation and increases resilience of the market to naturally respond to changing conditions [44].

5.1.2.3 Benefits to the Consumer

In this context, a consumer is defined as the end user of electricity in the traditional market configuration. This includes residential homes and industrial buildings.

Elimination of static, long-term agreements with a single provider allows consumers to freely choose the best offer for energy. This creates true market competition between providers offering the lowest price or best quality product (e.g. highest renewable percentage). Micro-SLAs increase the buying power of a single consumer by obliging providers to respond to consumer preferences. For instance, a small, independent energy provider produces 100% green energy. Consumers choose to immediately migrate to this provider. This sends a signal that consumers prefer this energy option. Other providers are obliged to offer similar products to entice consumers to return.

Micro-SLA also increases the selling power of consumers with local energy production (e.g. solar panels on a residential roof). Instead of the current, obligation to sell overcapacity back to a single provider for a fixed price[6], prosumers (i.e. consumers who may also produce) offer energy to the marketplace at large. Pro-sumers may sell overcapacity to a selection of providers or even to other (local) consumers. In this context, the consumer assumes the role of provider and attempts to find the highest bidder. The possibility of reselling energy may further motivate consumers to reduce consumption during peak periods, because energy sales during these periods are likely to fetch a higher price.

[6] In the Netherlands, it is currently neither technically nor administratively possible to sell self-generated energy to more than one energy provider [116].

Producing consumers (pro-sumers) may also form groups to satisfy energy needs independently of large, commercial producers. Such groups form a Virtual Power Plant (VPP). A VPP is a collection of small-scale energy providers and consumers that satisfy each others' energy needs, independent of the rest of the energy grid [47, 118].

5.1.2.4 Benefits to the Producer

In this context, a producer is defined as the generator of electricity in the traditional market configuration. This includes, centralized power plants such as coal, gas and nuclear plants, as well as off-shore wind farms and concentrated solar farms.

One benefit to the producer is reduction of overcapacity. When consumers are able to react to price fluctuations, consumption during peak periods is reduced and less overcapacity is required [17, 138]. This reduces wasted energy resources and increases profit margins.

An additional long-term benefit that stems from reduction of overcapacity is a decrease in capital expenditure and operating expense [161]. Consistent, long-term reduction of overcapacity requirements equates to reduced investment in production capability. For instance, an additional turbine that produces 20% of total capacity will not be built and an existing one may be decommissioned earlier than expected.

Smaller producers with 100% green, intermittent sources compete in a market place with short-term contracts. In the traditional market, an energy producer with intermittent sources (e.g. wind) also requires a portfolio of stable energy sources (e.g. coal) to guarantee its consumers continuous power [129]. This requires significant investment and thus smaller producers sell to other retailers rather than directly to the end consumer. With micro-agreements the owner of a wind farm sells directly to customers dependent on how much energy is being produced due to weather conditions. When production is high, the client base is increased dynamically and vice versa. This increases revenue and reduces wasted potential energy. Furthermore, allowing consumers to react to changes in energy prices provides small, specialized producers with more market power [40].

Another benefit to producers is direct access to user preferences. With micro-agreements, consumers 'vote' (i.e. with pocketbooks) for energy preferences in real-time. Producers follow consumer trends and react by investing accordingly. Typically such data are collected through limited surveys or analyzing trends over longer periods, such as years or decades.

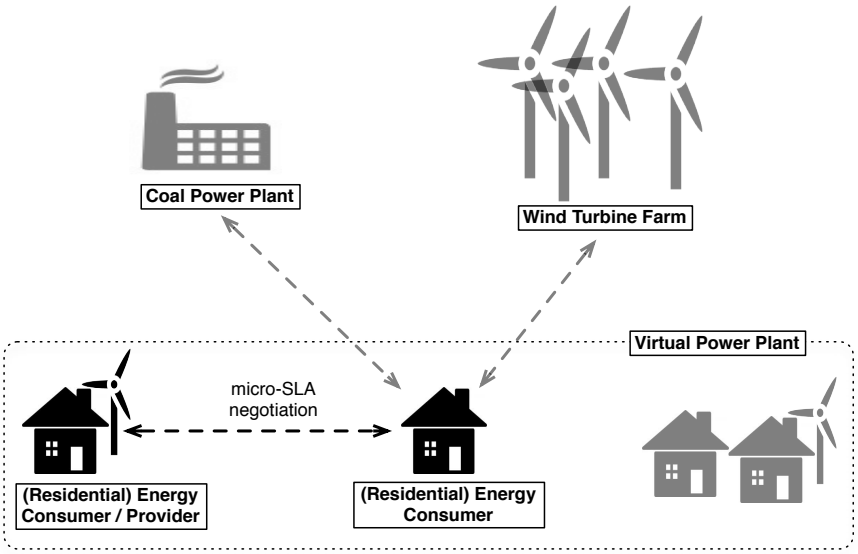

Figure 5.1: A future energy market with multiple providers and consumers.

5.1.3 Energy Negotiation Scenario

This section illustrates an implementation of an energy negotiation scenario using the negotiation and monitoring framework discussed in Chapters 3 and 4. This scenario is positioned within the context of the future energy market introduced above. Figure 5.1 illustrates the relationship between multiple energy providers and consumers in an example future energy market. In this figure, a (residential) energy consumer negotiates micro-SLAs with multiple providers. Providers include a centralized, coal power plant, a centralized, wind turbine farm and several decentralized (residential) producers (e.g. small wind turbines or solar panels). Optionally, small-scale providers and consumers group together to form a Virtual Power Plant (VPP).

In this scenario, a single energy provider and a single energy consumer negotiate a micro-SLA. Both participants are represented by their own individual software agents in the AgentScape middleware, presented in Chapter 2.3.2. The WSAN Service provides service discovery, negotiation, agreement creation and, usage monitoring. Source code is found in the Appendix and is available upon request.

5.1.3.1 Energy Provider Agent

The energy provider agent presents available services in a template, using WS-Agreement XML. Figure 5.2 illustrates this template. The template begins with context information, including the name of the template and provider agent. In addition, this template lists the available services: *EnergyService*. This service has two description terms: *EnergySource* and *EnergyPrice*.

This template also includes *Creation Constraints* (see Section 3.2.2). These constraints define the acceptable values of each negotiable term. In this instance, the value of EnergySource is constrained to those discrete values listed in the enumeration type. These values include `NUCLEAR`, `COAL`, `GAS`, `WIND`, `SOLAR`, `HYDRO`, `BIOFUELS` and `GEOTHERMAL`. The value of EnergyPrice must be a double (e.g. real number) that is greater than or equal to zero (e.g. no negative values allowed).

In addition to one or more service templates, the energy provider agent requires a negotiation policy. The policy provides the agent with enough information to negotiate autonomously. Each policy is domain and context dependent and is completely customizable to an individual agent. Policies encapsulate both preferences (e.g. priorities, ideal price) and strategies (e.g. if too high, decrease by 10%). Appendix A contains JAVA source detailing an example policy (see Figures A.1 through A.5). Note that this example policy is intentionally simplified for the purpose of illustration.

Figure A.1 in the Appendix indicates how a provider agent may specify the minimum prices for each available energy source. These prices may be based on current market prices or on historical data. In addition, a provider agent may specify a range of values for each energy source. For example, in addition to minimum prices, a provider also specifies ideal prices (e.g. 50% higher than minimum prices). If an offer is under the minimum price, one strategy is followed. If an offer is above the minimum but below the ideal price, a different strategy is followed. A provider agent thus customizes its response appropriately.

In addition to specifying personal preferences, a policy also includes several obligatory methods for the negotiation process. One of these methods is a boolean (e.g. acceptable or not) test for evaluating incoming negotiation offers: `evaluateNegotiationOffer()`. An example of this method is presented in Figure A.2 in the Appendix. First, all negotiation terms are extracted from the offer. If the Energy Service is found, then each term is extracted and evaluated, in turn. In this example, the *WIND* Energy Source is extracted and the offered price is evaluated. If the price is acceptable, and all other included terms are acceptable, then the method returns `true`. However, if the price of WIND, or another term, is unacceptable, the method returns `false`

```
<Template TemplateId="4">
  <Name>EnergyTemplate</Name>
  <Context>
    <ServiceProvider>AgreementResponder</ServiceProvider>
    <TemplateId>4</TemplateId>
    <TemplateName>EnergyTemplate</TemplateName>
  </Context>
  <Terms>
    <All>
      <ServiceDescriptionTerm Name="RESOURCE_SDT"
          ServiceName="EnergyService">
        <Energy>
          <EnergySource/>
          <EnergyPrice>0.0</EnergyPrice>
        </Energy>
      </ServiceDescriptionTerm>
    </All>
  </Terms>
  <CreationConstraints>
    <Item Name="EnergyService_EnergySource">
      <Location>AgreementOffer/Terms/All/ServiceDescriptionTerm/
          Energy/EnergySource</Location>
      <ItemConstraint>
        <simpleType>
          <restriction base="string">
            <enumeration value="NUCLEAR" />
            <enumeration value="COAL" />
            <enumeration value="GAS" />
            <enumeration value="WIND" />
            <enumeration value="SOLAR" />
            <enumeration value="HYDRO" />
            <enumeration value="BIOFUELS" />
            <enumeration value="GEOTHERMAL" />
          </restriction>
        </simpleType>
      </ItemConstraint>
    </Item>
    <Item Name="EnergyService_EnergyPrice">
      <Location>AgreementOffer/Terms/All/ServiceDescriptionTerm/
          Energy/EnergyPrice</Location>
      <ItemConstraint>
        <simpleType>
          <restriction base="double">
            <minInclusive value="0.0" />
          </restriction>
        </simpleType>
      </ItemConstraint>
    </Item>
  </CreationConstraints>
</Template>
```

Figure 5.2: WSAN XML Template advertising energy services.

and the offer is unacceptable. The result of this method does not necessarily determine the outcome of negotiation. This method only evaluates if an offer is currently acceptable or if further analysis and negotiation are required.

If an offer is not acceptable, the provider is able to take several actions. One possible action is that the provider terminates negotiations. Another action may be to reject the offer, optionally, providing a reason for rejection. This rejection may prompt the consumer to propose a more acceptable offer. An additional action the provider may choose to take is to create a counter-offer.

A method is included in the policy that specifies how counter-offers are created: `createCounterOffer()`. An example of this method is presented in Figure A.3 in the Appendix. First, all negotiable terms are extracted from the offer. If the Energy Service is found, then each term is extracted and evaluated, in turn. In this example, the unacceptable *WIND* Energy Source price is detected. A new, acceptable value is calculated according to the provider agent's personal strategy. In this example, the strategy dictates that the new WIND value is 10% above the minimum value (see Figure A.1 in the Appendix). This new value is inserted into a new offer document and returned to the consumer agent.

In addition to these two methods, several helper methods are provided by the negotiation framework. These methods handle incoming offers, based on the `state` of an offer. The state of an offer determines the possible actions that are taken in response. The four possible offer states specified by WS-Agreement Negotiation (see Section 3.3.3) are: `ACCEPTABLE`, `ADVISORY`, `REJECTED` and `SOLICITED`. The corresponding helper methods are: `processAcceptableOffer()`, `processAdvisoryOffer()`, `processRejectedOffer()` and `processSolicitedOffer()`. Figures A.4 and A.5 in the Appendix provide an overview of these methods.

Agents handle incoming offers in the ACCEPTABLE or ADVISORY state following similar processes. First, a response offer is created. Regardless of the outcome, this document will carry the response message to the consumer agent. Then the offer is evaluated using the `evaluateNegotiationOffer()` method (see Figure A.2 in the Appendix). If acceptable, the offer is returned without changes. However, if the offer is unacceptable, a counter-offer is created using the `createCounterOffer()` method (see Figure A.3 in the Appendix). Finally, the response is returned to the consumer agent.

Incoming offers in the REJECTED state are handled separately. All responses are possible, but the specific actions taken depend on an agent's personal strategy. For instance, a provider may terminate negotiation. In contrast, a provider may exploit the rollback support of WS-Agreement Negotiation to create a new counter-offer with better terms.

Incoming offers in the SOLICITED state are an exceptional case. This state limits the possible responses, as it is designed to illicit a 'yes or no' decision from the counter-agent. When an offer is received in this state no counter-offer is possible. Figure A.5 in the Appendix illustrates a method for handling such situations. First, a response offer is created. This document will be identical to the incoming offer with the exception of the state. Depending on the outcome of this method, the state will either be changed to ACCEPTABLE or REJECTED. Then the offer is evaluated using the `evaluateNegotiationOffer()` method (see Figure A.2 in the Appendix). Based on the outcome of this evaluation, the state is set accordingly to either ACCEPTABLE or REJECTED. Finally, the response is returned to the consumer agent.

5.1.3.2 Energy Consumer Agent

The energy consumer agent uses the WSAN Service to discover suitable templates from energy providing agents. Once a template is found, the consumer initiates negotiation with the provider. Similar to the provider agent described in the previous section, decisions made by the consumer during negotiation are governed by the consumer's personal negotiation policy. The policy provides the agent with enough information such that it is able to make negotiation decisions autonomously. Each policy is domain and context dependent and is completely customizable to an individual agent. Policies encapsulate both preferences and strategies. Figures A.6 and A.7 in the Appendix illustrate an example consumer policy.

Figure A.6 in the Appendix indicates how a consumer agent may specify the maximum price it is willing to pay for a particular energy source. These prices may be based on personal preferences. For example, one consumer may be willing to pay more for renewable sources and less for nonrenewable sources. Similar to the provider policy described in the previous section, a consumer agent may specify ranges of values for each energy source.

In addition to personal preferences, a policy also includes several methods for the negotiation process. These methods include those already discussed in the previous section, including `evaluateNegotiationOffer()`, `createCounterOffer()` and methods to process all offer states. Depending on which agent initiates negotiation, an additional method is required: `createNegotiationOffer()`. An example of this method is presented in Figure A.7 in the Appendix.

This method creates an initial offer, based on a template. Offer creation follows a personal strategy. For example, the initial offer begins with the minimum price. Additionally, the initial offer, as with all subsequent offers, is validated against the creation constraints of the relevant template. Figure 5.2

provides an example of creation constraints. The chosen values of the initial terms must comply with these creation constraints. For instance, the chosen value of `EnergySource` must be equal to one of the values listed in the creation constraints, such as `NUCLEAR` or `COAL`. If the creation constraints are violated, the offer is rejected immediately, without further analysis.

If the creation constraints are not violated, then the `createNegotiationOffer()` method proceeds as follows. First, an empty offer document is created. Then the unique session identifier (received from the WSAN Service) is added to the new document. Then the negotiation context is prepared. This includes setting the roles of each agent, setting the initial state (e.g. ADVISORY) and setting the `counterOfferTo` field. The initial offer has no previous offer on which it is based. Therefore, the counterOfferTo field refers to the original template. In subsequent offers, this field is automatically set to the chosen previous offer.

Once the context is set, the negotiation terms are filled in. The selection of these terms follows a consumer agent's personal negotiation strategy (e.g. begin 10% of the maximum price). Once the terms are filled in and validated against the creation constraints, the initial offer is sent to the provider agent.

Subsequent counter-offers are created using `createCounterOffer()`. The consumer agent's version of this method is similar to the provider agent's method (see Figure A.3 in the Appendix). The consumer responds to unacceptable negotiation terms following a personal strategy. For example, a counter-offer contains terms 10% higher or lower than the unacceptable terms from the previously received offer.

5.1.3.3 Energy Negotiation

The negotiation of energy services is a process of exchanging documents following the WS-Agreement Negotiation specification (`NPS-1`). These documents contain (counter) offers with various combinations of negotiation terms. Figure 5.3 presents an overview of the negotiation process. In this particular scenario, the energy consumer agent initiates negotiation. Based on the energy template presented in Figure 5.2, the consumer agent creates the initial negotiation offer. Appendix A contains JAVA source illustrating the message exchange between agents, including this initial offer in Figure A.8 in the Appendix. Source code is also available upon request.

The initial offer contains a unique identifier: `initiator-1`. This format implies which agent created the offer and the number (and ordering) of the offer. In addition, each offer carries the unique session identifier. As the session identifier is not part of the official WS-Agreement Negotiation (`NPS-1`) specification, it is not included in the offer document itself. Rather, the session

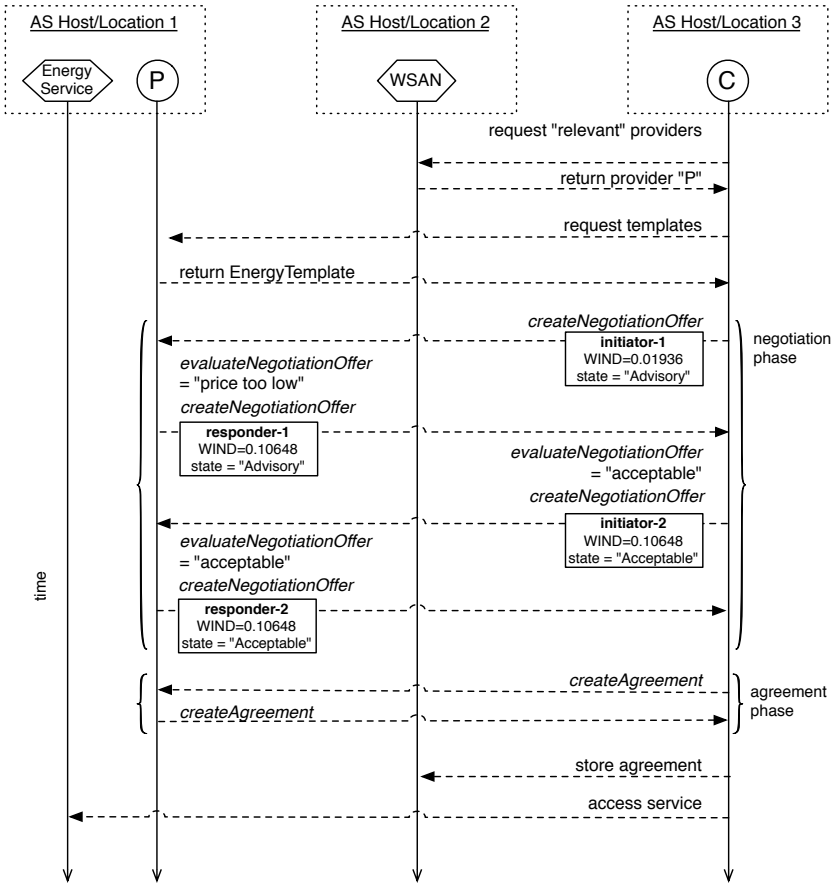

Figure 5.3: Energy negotiation scenario using WS-Agreement Negotiation (`NPS-1`) protocol in AgentScape. Source code of all offers and evaluation methods is available in the Appendix and upon request.

5

identifier (from `NPS-2`) is added to the encapsulating message file that is sent between agents. The combination of a session identifier and offer identifier makes it possible to uniquely identify each offer and its creator.

In addition to the identification information, the offer contains two context sections: `Context` and `NegotiationOfferContext`. These sections correspond to the Agreement Layer and the Negotiation Layer, respectively. The first context section identifies the roles of each agent in the Agreement Layer and the original template: `EnergyTemplate`. The provider uses this information to assist validation of the included terms against the original creation constraints.

The second context section identifies the creator and state of this particular offer. In addition, this section contains the identifier of the previous offer.

This information assists possible rollbacks or building the negotiation tree (see Figure 3.7). In this example, the previous offer is the original template.

The selected terms and corresponding values are listed in the `Terms` section of the offer. The initial price for `WIND` is `0.01936`, which is equal to the 10% of the maximum price, as specified by the consumer agent's personal policy.

Upon receiving the initial offer, `initiator-1`, the energy provider agent evaluates the terms and responds with a counter-offer,`responder-1`. Figure A.9 in the Appendix depicts this counter-offer. This document contains the same sections as the first offer with minor, but significant changes. First, the `OfferId` is updated to reflect the creator and sequence. In the negotiation context, the creator is updated and the `CounterOfferTo` is updated to reflect the previous offer identifier.

The price of WIND is changed. The proposed price is below the energy provider's specified minium (0.0968). Following the energy provider's strategy, the new price is 10% above this minimum: `0.10648`.

Upon receiving the provider's first counter-offer, `responder-1`, the energy consumer agent evaluates the terms. The proposed price of WIND is less than the maximum value specified by the consumer's personal policy. The offer is therefore evaluated as acceptable. At this point, the consumer may propose a new price, lower than the provider's suggested price. Instead, the consumer indicates that this price is acceptable returning the offer largely intact. Figure A.10 in the Appendix depicts this new document, `initiator-2`. The terms do not change. The only changes that occur are an updated identifier and context information, including a new state: `Acceptable`.

Upon receiving the consumer's reply, `initiator-2`, the energy provider agent evaluates the terms, once again. Although the terms are identical to those proposed by the provider, itself, the provider's policy may have changed in the interim. For example, changes in actual supply may increase the price dynamically. If the terms are still acceptable and the provider does not wish to suggest (higher) prices, the provider accepts the offer. At this point, the provider returns the identical offer, with only minor changes to the context information. This new document, `responder-2`, is presented in Figure A.11 in the Appendix.

Receipt of this document indicates to the energy consumer agent that an acceptable offer has been negotiated. The consumer may now exit the Negotiation Layer and initiate agreement creation in the Agreement Layer. As the offer is already acceptable to both agents, there is a high likelihood of successful agreement creation. If successful, a copy of the agreement is stored at the WSAN Service for administrative and monitoring purposes.

5.1.3.4 Energy Monitoring

In this scenario, monitoring is applied to several areas of the example future energy market. First, passive monitoring monitors SLA compliance between the energy consumer and provider. Section 4.2 discusses passive monitoring in more detail. At regular intervals, both agents commit to continue the service for the following interval. This commitment implies the satisfaction of both agents with the quality of service.

In addition, monitoring is applied to energy usage within Virtual Power Plants (VPPs). Monitoring is applied within this group to measure and record energy production and usage between members. In this context, monitoring serves dual purposes: (1) violation detection and (2) accounting. The monitor detects SLA violations between consumers and providers. Agents store monitoring data for accounting purposes. Accounting determines how much energy is consumed by a particular member and which providers are (financially) compensated.

5.1.3.5 Energy Scenario Conclusion

The scenario presented in the previous sections illustrates how a multiagent framework is applied to negotiate and monitor SLAs in the energy domain. Agents representing an energy consumer and provider negotiate the terms of an SLA. Each agent evaluates offers and proposes (counter) offers following a personal negotiation policy. A policy specifies an individual user's preferences, priorities and actions in a given situation. In this particular implementation, policies are specified using the Java programming language. However, the structure and principles of the policy are language independent.

The negotiation process detailed in this scenario uses the WS-Agreement Negotiation (`NPS-1`) protocol to structure the negotiation dialogue. Offers are created, analyzed and responded to following this protocol. Each offer is formulated in the WS-Agreement Negotiation language. This scenario shows how this protocol encapsulates a given agent's preferences and guides the negotiation process to conclusion. In this scenario, several rounds of negotiation are necessary before an acceptable offer is reached. In contrast to WS-Agreement Negotiation, a protocol that does not support multiround negotiation would fail to reach a successful agreement in this particular scenario.

This scenario depicts a single negotiation process. Note that this process is repeated each time an active micro-SLA expires. In a future energy market in which the lifetime of micro-SLAs is 1 hour, this process is thus repeated hourly. The multiagent negotiation and monitoring framework enables users to automatically manage this otherwise tedious process.

5.1.4 Discussion

In recent years, much has been invested in green energy sources. These resources are significantly underutilized. Green energy supply does not efficiently reach consumers when they need it. Future energy markets require several changes to increase this efficiency using demand side management (DSM). First, micro agreements replace long term contracts. Micro agreements allow consumers to quickly react to changes in the market, such as price and availability. Second, real-time pricing (RTP) provides consumers with an incentive to react to market signals by shifting demand. For instance, reducing consumption when wind power generation is scarce and increasing consumption when it is abundant.

Studies show the effect of RTP at encouraging DSM in energy markets. In future energy markets, autonomous agents further increase this effectiveness. Agents represent energy providers and consumers to negotiate micro-agreements. This new market structure reduces wasted energy (e.g. overcapacity), lowers prices and increases utilization of green resources.

This market also enables consumer participation. Consumers can actively favor green energy sources and reward investment in new renewable sources. Groups of consumers can form independent VPPs to supply energy demand with distributed sources, independent of the energy grid.

5.2 Dynamic Services in the Cloud

Cloud computing [6] provides the illusion of unbounded online resources, such as CPU or storage capacity. Companies that offer these resources are referred to as Cloud Service Providers (CSP). The Cloud is sometimes also called *elastic* since customers are able to easily increase or decrease resource usage, such as the amount of computing power, rented from a CSP.

Similar Cloud services are offered through a number of CSPs that compete on price and service levels. Several of these CSPs also offer numerous options to customers who customize services based on metrics such as price, Quality of Service (QoS), reputation and location. Note that most of these metrics are dynamic, i.e. they change continuously. For example, some CSPs, such as Amazon Web Services spot pricing, offer dynamic pricing. This enables that the price of resources changes constantly, which reflects underlying factors, such as Cloud utilization, fluctuating energy prices or consumer demand [4, 130].

In this environment, a consumer of Cloud services faces several challenges. First, to obtain the desired initial configuration of Cloud resources, a consumer will evaluate prices and configuration options (QoS levels, location, etc.) of all

available CSPs. The task of finding the ideal configuration is further complicated as more CSPs implement dynamic pricing. When a consumer chooses the configuration that is currently the most appropriate, a better (cheaper) configuration may become available soon thereafter. Therefore, a consumer must periodically reevaluate configurations at all available CSPs. If a consumer chooses to move from his/her current CSP to a different CSP with a more suitable configuration, the consumer is then faced with the challenge of migration. Due to a lack of interoperability of CSPs and the tendency towards vendor lock-in, changing CSPs is not a trivial task. Finally, once a consumer chooses a CSP, the consumer must continually monitor the service to detect violations to the service agreement. Moreover, the consumer must also provide evidence, for example in the from of an audit trail, that a violation has actually taken place.

To assist a consumer with these challenges, this section introduces an Intelligent Cloud Resource Allocation Service (ICRAS). ICRAS supports the consumer throughout the lifecycle of a Cloud service. This includes, (1) discovering all available resource configurations, (2) choosing the desired configuration, (3) negotiating a service agreement with the CSP, (4) assisting in the migration of services between CSPs and (5) securely monitoring the service agreement for violations.

ICRAS aggregates information describing the available services from multiple CSPs, including current price, availability, Quality of Service guarantees, location and reputation. When a consumer requires resources, it contacts ICRAS with a description of the computing needs. ICRAS then matches the resource request to the most appropriate configuration of Cloud resources from the CSPs. ICRAS facilitates the negotiation of the necessary Service Level Agreements (SLA) with the CSPs on behalf of the consumer and assists in the migration process.

ICRAS then monitors the services during the lifetime of the SLA to ensure that there are no agreement violations. If violations are detected, corrective action is taken. Service monitoring uses secure modules at both the consumer and provider. Further steps are taken to generate an audit log of service message. Using several cryptographic protocols, this audit log guarantees integrity and nonrepudiation of service messages.

The following sections describe the application of autonomous negotiation and monitoring in the Cloud. First, Section 5.2.1 describes the core concepts of Cloud resource allocation, including the challenges of standardization and dynamic pricing. Then, Section 5.2.2 then describes ICRAS in more detail, including an overview of the architecture and protocol. Finally, Section 5.2.3

demonstrates the applicability of ICRAS for a use case based on a prototype implementation.

5.2.1 Cloud Resource Allocation

The Cloud refers to hardware and software resources available across the Internet [6]. Cloud services are roughly categorized as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). This categorization is based on the complexity of the service, from raw compute resources, such as storage or processing power, to refined software services, such as databases or other applications.

The following sections focus mainly on the first of these categories: IaaS. An example of IaaS are the resources offered by Amazon Web Services (AWS)[7]. AWS offers consumers a computing 'instance' that includes customizable attributes, such as processing power, memory and disk space and operating system (e.g. Windows). Consumers create as many instances as needed. AWS offers both a web interface for human access, as well as a scriptable, Java-based Application Programming Interface (API) for automated access.

The Cloud computing model allows end users to rent computing infrastructure as needed, rather than requiring them to purchase resources outright. Moreover, consumers are able to quickly and simply adjust the size of Cloud resources, depending on current computing requirements. Consumers use these services following a pay-as-you-go model, only paying for the specific amount of time or level of service they consume.

Much research into efficient use of Cloud resources focuses on increasing utility of the CSP. For instance, techniques are proposed for load-balancing techniques aimed at reducing energy costs [9] or dynamic pricing models that maximize revenue [130]. In contrast, this section proposes a service to maximize utility from the perspective of the consumer. With this service, a consumer finds the most appropriate balance between low cost and high quality.

In some cases, automation assists in the process of load-balancing or scale resource allocation based on utilization[8]. For example, a Cloud consumer may create an agreement that specifies the upper and lower bounds of resource allocation (e.g. at least 5 instances, but no more than 100). When utilization of these resources exceeds a given threshold (e.g. 90% CPU utilization), new instances are automatically created. When utilization decreases, instances are automatically terminated.

[7] `http://aws.amazon.com`

[8] Amazon Auto Scaling. `https://aws.amazon.com/autoscaling/`

5.2.1.1 Standardization

Despite several exceptions, including those noted above, Cloud resource allocation is currently largely a manual process. Consumers manually compare, negotiate and allocate resources. This section proposes fully automating these tasks to (1) empower consumers to effectively utilize resources and (2) increase efficiency of resource allocation. However, several challenges limit automation of these tasks.

One challenge to automated Cloud resource allocation is the lack of standardization. Cloud resource offerings rely on *virtualization* [59]. (Hardware) virtualization simulates a computer environment on top of physical resources. The virtualization layer provides an abstraction between underlying hardware (e.g. processing and storage) and structured services (e.g. IaaS, PaaS or SaaS). In practice, a single physical machine simulates multiple virtual machines or multiple physical machines simulate a single virtual machine.

Offering Cloud resource consumers this abstraction from physical hardware requires several virtualization components, including (1) platform/API (2) hypervisor and (3) disk images. There are several alternatives for each of these components and compatibility between alternatives remains a challenge.

A Cloud *platform* is a management and control layer that enables deployment of Cloud resources. Essentially, this layer creates a virtualized service (e.g. a certain amount of processing and storage) and assigns this to physical hardware. Consumers access this layer either via a web-based interface or an automated API. Many competing platforms exist and are incompatible. Two emerging standards are OpenStack [119] and Eucalyptus [117].

Each Cloud platform supports one or more *hypervisors*. A hypervisor is software that simulates a virtual machine (e.g. a Cloud instance). This software is responsible for providing the abstraction between virtualized services and the underlying hardware. Many competing hypervisors exist, including VMware's ESX[9], Citrix's XenServer[10], Microsoft's Hyper-V[11] and the open source KVM[12]. In limited situations, it is possible to migrate virtual machines from one hypervisor to another [27]. However, compatibility between hypervisors remains a challenge.

Each hypervisor supports one or more *Virtual Disk Images (VDI)*. A VDI is an encapsulation of a virtual machine's data (e.g. disk storage). Many alternative VDI formats exist, such as VMware's VMDK, Citrix's Xen VHD and

[9] http://www.vmware.com/products/vsphere/esxi-and-esx/index.html
[10] http://www.citrix.com/products/xenserver/overview.html
[11] http://www.microsoft.com/en-us/server-cloud/hyper-v-server/default.aspx
[12] Kernel-based Virtual Machine. http://www.linux-kvm.org/

Table 5.1: Limited import/export functionality among leading CSPs.

CSP	*AWS*	*CloudSigma*	*MS Azure*	*Terremark*
import	VMware VMDK, Xen VHD, MS VHD	raw	MS VHD	**OVF**, VMware VMDK
export	AMI (snapshot)	raw	MS VHD	none

Microsoft's VHD. Most formats are only compatible with a single, specific hypervisor. As with the Cloud platform, attempts are made to standardize VDI format. One emerging standard is the Open Virtualization Format (OVF) [42]. This format includes both the user's data (e.g. disk image) as well as additional metadata that describes the necessary hardware, such as the type and number of processors or amount of memory.

Competing alternatives for Cloud platform, hypervisor and VDI and lack of widespread adoption of the emerging standards impedes compatibility between CSPs. This makes it challenging for a Cloud consumer to migrate services from one CSP to another. Two general approaches to migration are possible: online (i.e. live) and offline. Online migration of resources moves data from one CSP to another while keeping the services running and accessible. This form of migration is only possible in limited circumstances when both CSPs are running the same hypervisor [27]. Offline migration stops all services and preserves data and, optionally, additional state (e.g. running processes, memory stack). These data are then transferred to the target CSP and the services are started. Services may be unavailable for a period of time during migration.

An obstacle to offline migration is incompatibility between VDIs supported by each CSP. Table 5.1 shows the limited import/export functionality offered by leading CSPs. AWS supports several import formats, but offers no export format aside from a proprietary (and encrypted) disk image. MS Azure only supports its own native VDI for import and export. Terremark supports the emerging OVF standard for import, but offers no export options. Of these leading CSPs, CloudSigma is the only leading CSP that offers a nonproprietary export format. Other CSPs, including Rackspace, Go Grid, SoftLayer and GreenQloud offer no import or export options.

5.2.2 Intelligent Cloud Resource Allocation

The Intelligent Cloud Resource Allocation Service (ICRAS) requires an underlying architecture, consisting of three major components: 1) a consumer, 2) a

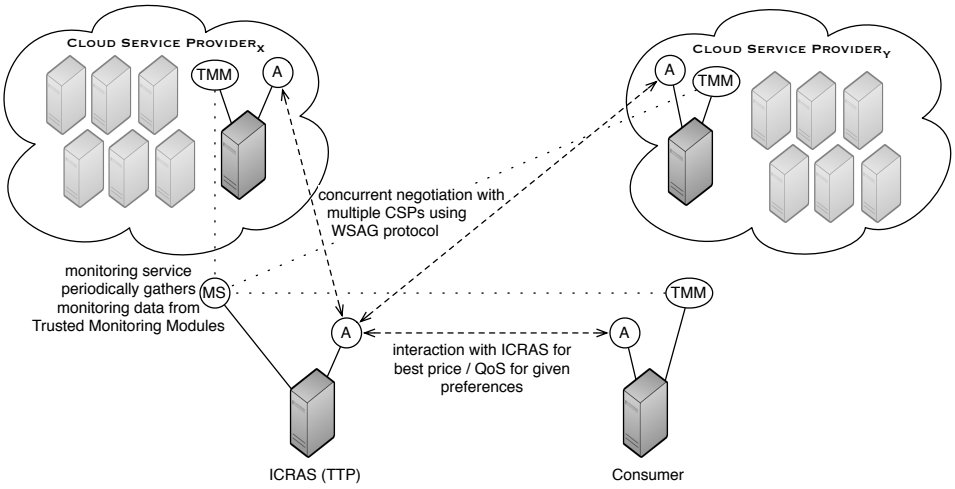

Figure 5.4: ICRAS architecture with a consumer negotiating with two competing CSPs

CSP and 3) an ICRAS agent. These elements represent the three roles in the marketplace, which may contain multiple instances of each. Furthermore, this architecture provides the mechanisms and protocols that enable these agents to communicate with one another and autonomously negotiate micro-SLAs. SLAs are negotiated and created following the WSAN specification. Figure 5.4 illustrates this architecture.

Consumer

Each consumer interacts directly with an ICRAS agent. A consumer specifies his/her requirements in an SLA offer. This document allows a consumer to specify 1) hard and 2) soft requirements, 3) priorities, 4) ranges of options, and 5) dependencies between requirements. For instance, a consumer requires 10 virtual servers with a combined CPU power of 20 GHz and a combined storage of 2 TB. Using the SLA notation, a consumer expresses that the CPU and storage requirements are strict, however, for a reduced price, the actual number or servers may change.

In addition to providing the initial resource requirements, a consumer is also responsible for updating these requirements. If resource requirements change, a consumer must inform an ICRAS agent of these changes. A change in requirements may occur for several reasons. First, based on current events or past experience, a consumer predicts increases or decreases in computing needs. For instance, online retailers receive more traffic leading up to the

holidays. Second, a change in business needs prompts an immediate reconfiguration of the resource requirements. For instance, a company decides to remove some legacy applications. Finally, a company's resource requirements change due to developments in the market, such as increased competition or lower consumer demand.

To enable such changes, a consumer monitors the level of activity on his/her Cloud resources and informs the ICRAS agent if a threshold is crossed and a new configuration is necessary.

The consumer must also host a Trusted Monitoring Module (TMM) (see Chapter 4). This module provides the ICRAS agent, acting as the de facto Trusted Third Party (TTP), access to relevant service metrics. The ICRAS agent thus accurately assess the user experience of the service. Passive monitoring is supported by extending the TMM to include the necessary cryptographic protocols.

CSP

To enable participation in the ICRAS architecture, a CSP must offer a compatible interface that is accessed by the ICRAS agent. This interface must support two main functions: negotiation and migration. For negotiation, a CSP must generate SLA templates. For this, a CSP requires access to internal information of its Cloud. This includes realtime pricing data, Cloud utilization and system health (QoS) information, if available. On the basis of this information a CSP generates SLA templates describing the available resources. Due to the dynamic nature of CSP resource availability and pricing, these SLA templates are updated regularly.

Upon request, the SLA templates are delivered to the ICRAS agent. When the ICRAS agent makes an offer, the CSP enters a negotiation session. The strategy that drives this negotiation is determined by the CSP negotiation policy. This policy includes functions for evaluating an offer, threshold values for acceptance or rejection of an offer and rules governing the creation of counter-offers.

To support data migration to and from its Cloud, the CSP interface must support the import and export of virtual disk images. After creating an SLA with a consumer, the CSP must support upload and import of the consumer's virtual disk images. Likewise, these virtual disk images are exported and downloaded upon request.

To support monitoring, the CSP must also host a Trusted Monitoring Module (TMM) that allows the ICRAS agent access to relevant service metrics, such as network latency. The TMM also includes support for the cryptographic protocols required for passive monitoring.

ICRAS agent

This section assumes that an ICRAS agent is maintained by an independent, Trusted Third Party (TTP). This service has no loyalty to a particular CSP and therefore operates fully on behalf of participating consumers. The ICRAS agent has five major responsibilities: 1) discover CSP resource offerings, 2) evaluate these offerings, 3) negotiate an SLA with a CSP on behalf of a consumer, 4) monitor the provisioning of the new Cloud resources to detect SLA violations and 5) assist in migration to the new CSP.

Discovery - The process begins when an ICRAS agent receives a resource request from a consumer. The agent then queries all CSPs for one or more SLA templates describing available resource offerings. This process is repeated at a regular interval to discover more appropriate configurations even after an SLA has been created. Depending on a consumer's preferences, he/she is notified if a new and more suitable configuration is discovered. Then the consumer can, optionally, renegotiate a new SLA.

Evaluation - Once received, the agent compares the CSP templates to the consumer's request. If a CSP is unable to provide the requested resources, this CSP is removed from consideration. The remaining templates are then evaluated and ordered using the preferences of the consumer. For instance, if a consumer specifies that price is the most important attribute, the remaining templates are arranged by price. Depending on a consumer's requirements, templates from multiple CSPs are selected for separate resource requirements. For instance, a consumer may allow two CSPs to handle processing and storage, separately, if this meets the price and QoS needs.

Negotiation - Once the best template has been selected, the ICRAS agent contacts the responsible CSP to begin negotiations. If multiple templates from competing CSPs are acceptable, these CSPs are contacted for simultaneous negotiations. If a negotiation session results in an offer that is acceptable by both a CSP and the ICRAS agent (according to a consumer's request), this is sent to the consumer for final approval. If acceptable, the consumer contacts the CSP directly to create a micro-SLA. A micro-SLA enables a consumer to migrate to a new configuration or renegotiate the current configuration if the opportunity arises.

Migration - Once a consumer decides to migrate, the consumer services are migrated to the new CSP. In the most straightforward case, migration involves stopping the cloud instances at the current CSP, converting these instances (e.g. disk images) to the format of the new CSP, transferring them to the new CSP and starting them again. The conversion process is not necessary if CSPs adopt the same industry standard, such as the Open Virtualization Format [42].

If services cannot be stopped during migration, live migration is required. In limited circumstances, live migration of cloud instances is possible if both CSPs are using the same virtualization layer [27]. However, the heterogeneity of current CSPs complicates the migration process.

Monitoring - The task of the ICRAS agent does not stop after SLA creation and service consumption. The ICRAS agent also assumes the role of Trusted Third Party (TTP) and monitors the service to detect SLA violations by either agent. Using TMMs at each agent, the ICRAS agent periodically measures service performance at both the source (CSP) and end user. The ICRAS agent uses a dedicated Monitor Service (MS) to monitor the SLA for QoS violations, such as slow network response [33]. If a violation is detected, agents are notified and corrective action is taken.

When using passive monitoring, the ICRAS agent acts as the mediator for conflicts that occur. As mediator, the agent requests audit logs from all agents. These logs are then analyzed to determine which, if any, agent has violated the SLA. The full mediation process is explained in detail in [84].

5.2.3 Cloud Negotiation Scenario

This section provides an example scenario to demonstrate the process of ICRAS mediated negotiation. This example involves two competing CSPs, a single ICRAS agent and a single consumer. For reasons of readability, service requests, SLA templates and offers are presented in generic format rather than the WS-Agreement XML format.

The ICRAS architecture is implemented using the AgentScape distributed middleware platform (see Section 2.3.2). Software (Java) agents represent the three major components: Consumer, ICRAS agent and CSP. Negotiation uses the WSAN Service that implements the WS-Agreement Negotiation specification (`NPS-1`) detailed in Chapter 3.

Two CSPs are chosen that fulfill the minimum standards of interoperability to support the example: Amazon Web Services[13] and CloudSigma[14]. On each of these CSPs, a server instance hosts a software agent running on AgentScape. Each agent uses the respective API to query price information and generate an SLA template describing each CSP's resource offerings.

An ICRAS agent runs on an instance of AgentScape on a local server. This agent collects templates from the agents running at each CSP. When the ICRAS agent has found the most suitable configuration, it is sent to the consumer agent, running on a separate instance of AgentScape on a separate local server. If a new CSP is chosen by the consumer, migration is assisted

[13] `http://aws.amazon.com/`
[14] `http://www.cloudsigma.com/`

```
          RESOURCE REQUEST
Num. of Servers = (10)
CPU GHz         = (1.5 - 3.0) | CD:C1, VI:V1
Storage  (GB)   = (2000 - *) | CD:D100, VI:V1
Traffic (GB)    = (1 - *) | CD:D1, VI:V1
Operating Sys.  = <Windows, Linux> | PC:YES
Availability    = [95 - 100) | CD:C2, VI:V1
Price (EUR)     = [0 - 1000) | CD:D2, VI:V1
```

Figure 5.5: Consumer generated resource request.

```
   SLA TEMPLATE CSPx              SLA TEMPLATE CSPy
Num. of Servers  = 100       Num. of Servers  = 50
CPU GHz          = 2.0       CPU GHz          = 3.0
Storage  (GB)    = 8000      Storage (GB)     = 4000
Traffic (GB)     = 1000      Traffic (GB)     = 500
Operating System = Linux     Operating System = {Windows OR Linux}
Availability (%) = 90        Availability (%) = 99
```

Figure 5.6: SLA template from two competing CSPs.

by the ICRAS agent. Virtual disk images are downloaded from the old CSP, converted to the target format using QEMU [53] and then uploaded to the new CSP.[15]

Step 1 - A consumer requires Cloud resources. A consumer specifies these needs using an SLA offer. Figure 5.5 summarizes this request. In this request, a consumer indicates that it needs 10 servers with CPU power between 1.5. and 3.0 GHz, at least 2 TB of storage and at least 1 GB of traffic. Furthermore, the consumer prefers the Windows OS, requires an availability of between 95 and 100 percent and a price below 1000 euro. This resource request is sent to the CSP.

Step 2 - The ICRAS agent receives the request of the consumer and queries all participating CSPs for SLA templates.

Step 3 - Each CSP receives the query and responds by sending SLA templates that describe the current resource offering to the ICRAS agent. If the templates have not yet been generated or are outdated, they are (re)generated at this point. The SLA template is generated following the WSAG specification. Figure 5.6 depicts example templates from two competing CSPs. In these templates, each CSP displays the current resource offering.

[15]Note that due to lack of standardization, a separate ad hoc solution for disk image migration is required for each unique pair of CSPs.

```
        SLA OFFER
Num. of Servers  = 10
CPU GHz          = 3.0
Storage  (GB)    = 3000
Traffic (GB)     = 10
Operating System = Windows
Availability (%) = 99
Price (EUR)      = 500
```

Figure 5.7: ICRAS agent generated offer.

Step 4 - Upon receiving the templates, the ICRAS agent evaluates each template using the consumer's request. If a template is unable to meet the requirements, it is immediately removed from consideration. In Figure 5.6, the template from CSP_x is removed because the availability offering is outside of the range specified by the consumer. In the case that more than one template remain after the first selection, the ICRAS agent evaluates them again to determine the most appropriate option. This evaluation is done by comparing key attributes, such as CPU or Availability.

Step 5 - At this point, the ICRAS agent has selected the best matching CSP. The ICRAS agent generates an initial SLA offer (see Figure 5.7). The ICRAS agent then contacts the selected CSP to begin negotiations. Following the WSAN specification, the negotiation consists of rounds of offers and counter-offers. If no mutually acceptable offer is found, negotiation terminates and the ICRAS agent selects a different CSP. However, in the event that a mutually acceptable offer is found, this offer is sent on to the consumer.

Step 6 - Once the consumer receives the offer, it re-evaluates the offering and, if acceptable, contacts the CSP directly to create a micro-SLA. After the SLA has been created, the service is accessed.

Step 7 - Upon successful creation of an SLA, the consumer migrates his/her services to the new CSP. This involves converting the virtual disk images to the format of new CSP and then transferring these images to the new CSP.

Step 8 - Upon successful creation of an SLA, the ICRAS agent takes on the new task of monitoring the service on behalf of the consumer. Monitoring is done by periodically measuring key service metrics and storing the result. If a violation is detected (e.g. Availability is less than promised), the consumer is notified and corrective action (e.g. fines, credits) is taken. In addition to SLA monitoring, the ICRAS agent also periodically requests and evaluates SLA templates from all CSPs. If a new offering is more appropriate than the current one, the consumer is notified and migration begins.

5.2.4 Discussion

The previous sections demonstrate automated negotiation and monitoring in the Cloud environment. The Intelligent Cloud Resource Allocation Service (ICRAS) (1) maximizes the utility of the consumer, (2) supports the consumer throughout the lifecycle of a Cloud service, (3) utilizes micro agreements in order to quickly react to changes in the Cloud service market (e.g. a lower price from a competing CSP), and (4) provides monitoring of the SLA which results in an audit trail that provides nonrepudiation and integrity.

The full potential of ICRAS is limited by lack of standardization between CSPs. If widely adopted, the Cloud platform, hypervisor and VDI standards, such as OVF, will make data and service migration between CSPs more straightforward. However, the main obstacle to adoption is vendor lock-in [180]. CSPs have little incentive to make the process of service migration possible, let alone straightforward; therefore, migrating away from a CSP remains a difficult task. A consumer does not always have the option to export or download a virtual disk images from a CSP. This means, once a consumer has migrated to a particular CSP, the cost and hassle of leaving that CSP prohibits them from doing so, even if a better configuration is found at a different CSP. Note that complete state-full migration, i.e., where a snapshot of a running image is migrated and the state of the newly migrated image is updated, is still an open research question. The discussed solution would only preserve the state until the snapshot is made, so some state is lost (e.g. when the image is migrating).

Finally, wider adoption of dynamic pricing in Clouds is needed to allow users to react to changes in real market forces, including Cloud utilization. Some providers offer dynamic pricing models to reflect the actual fluctuation of resource supply and demand. Dynamic pricing is beneficial to both consumers and providers of Cloud resources. Consumers shift demand to cheaper time slots, such as evening or weekend processing, to save on costs. CSPs take advantage of demand shifting to lower costs during peak periods. For instance, a CSP reduces the cost of cooling a data center at noon on a hot day by making it cheaper to use the data center at night.

Cloud computing was originally envisioned as a utility, similar to the electricity grid, where users simply plug in to meet computing needs. To enable this vision, more standardization and openness is required in the Cloud interface and data format.

The incompatibility of CSPs as discussed above greatly limits the ability to evaluate ICRAS. Preferably, an (exhaustive) evaluation tests and compares various metrics, such as migration delay and negotiation success rate. However, vendor lock-in of data formats prevents this. Section 5.2.3 describes

a scenario with two CSPs. These CSPs are specifically chosen for (limited) interoperability. However, even with these two carefully chosen CSPs, the experiment only functions in one direction. Migration is possible from CSP_1 to CSP_2 but not in the other direction. CSPs must adopt open standards, as discussed above, before more extensive evaluation of ICRAS is possible.

5.3 Conclusion

The use cases presented in this chapter demonstrate the applicability of the integrated negotiation and monitoring framework. Consumers and providers access framework mechanisms throughout the lifecycle of an SLA. These mechanisms assist users in service discovery, negotiation, agreement, provisioning and monitoring. This framework provides a structured and trusted platform for use in dynamic, open environments. This structure reduces the complexity of such environments and presents users with an effective means to utilize the underlying resources.

In the Smart Energy Grid, the automation framework enables users to quickly react to changes in the environment, such as green resource availability. Consumers are empowered to shift demand to increase green resource utilization. Providers reduce overcapacity and the correlated costs thereof.

In the Cloud, the framework reduces the complexity of incompatible, competing CSPs. The consumer is presented with a unified interface where resource preferences are entered and suitable Cloud resource configurations are returned. The framework provides trust in the underlying resources by providing transparency in service compliance. SLA violations are detected and penalized.

The two use cases are technically challenging and socially relevant. The dynamic nature of the underlying resources may result in reduced efficiency. Automation frameworks are needed to manage the underlying complexity and increase usability of these environments. Increasing the usability of either of these environments has direct social impact. For the Smart Energy Grid, increasing the utilization of renewable resources leads to less waste and pollution. For the Cloud, increasing the usability of resources enables wider adoption of Cloud services, reduction of unnecessary costs and increased efficiency of Cloud resource allocation.

CHAPTER 6

Conclusion

The future brings opportunity. Large scale, distributed, digital environments will provide access to vast amounts of knowledge and resources, creating new possibilities. Such environments are inherently dynamic and untrusted. Within these environments, software systems will provide unprecedented support for daily life. One such system is the Smart Energy Grid [61] that is being designed to increase sustainability and decrease reliance on fossil fuels. This system enables communities to take responsibility for their own production. Producers and consumers negotiate SLAs that specify which energy is provided and consumed. Determining if an SLA is violated requires distributed monitoring.

This dissertation presents a MAS framework for automated negotiation and monitoring in dynamic, distributed, open environments. Software agents represent (human) providers and consumers in a digital marketplace. Through a process of exchanging messages (e.g. offers, counter-offers), agents together search for a mutually acceptable agreement (e.g. service, price quality). A negotiation protocol defines the negotiation objects (i.e. offers), language and rules governing interaction. For multiagent negotiation in open environments, in which agents dynamically change roles (e.g. pro-sumer), protocols must be

flexible and symmetric. This dissertation presents the WS-Agreement Negotiation protocol[1] with extensions for open environments. This protocol is experimentally validated in the AgentScape middleware.

Open environments present challenges regarding security, trust and privacy. No single authority has complete control over an open environment and no single authority governs the actions of all participants (i.e. agents). Therefore, additional mechanisms are required to ensure security, privacy and promote trust between participants. Automated monitoring mechanisms using a Trusted Third Party (TTP) address issues of security and thus support negotiation in open environments. This dissertation presents a self-adaptive monitoring approach that (1) offers monitoring assurance that agreements are honored, (2) builds a secure audit log of agreement compliance, (3) performs measurements while safeguarding privacy of (sensitive) data, (4) dynamically reacts to changes in risk and (5) enables trust-building between consumers and providers. This monitoring approach is experimentally validated in the AgentScape middleware.

6.1 Research Questions Revisited

The overarching goal of this research is bringing the benefits of automation negotiation technology to open environments. The complex nature of open environments impedes manual negotiation and leads to inefficiencies. Automated negotiation can reduce manual workload and increase efficiency (e.g. matching supply and demand). The objectives of this research are first to gain understanding of the challenges of negotiation in open environments, and then to provide structure to support automated negotiation in these environments, such as languages, protocols and mechanisms.

Section 1.2.1 introduces the following research questions. Answering these questions provides the knowledge required to achieve the research objectives. This section addresses each question with the results of this research. First, this section addresses the subquestions, `RQ1` and `RQ2`, related to negotiation. Then it addresses the subquestions, `RQ3` and `RQ4`, related to monitoring. Finally, this section revisits the general research question.

`RQ1` Can protocols be designed to support natural negotiation dialogue between agents?

`RQ2` Can mechanisms be designed to facilitate reliable, secured negotiation?

[1] Now an official Open Grid Forum standard. Version 1.0 available at: `http://www.gridforum.org/Public_Comment_Docs/Documents/2011-03/WS-Agreement-Negotiation+v1.0.pdf`

Chapter 3 presents a negotiation protocol that supports natural negotiation dialogue between agents. The protocol enables bidirectional exchange of offers and counter-offers. Limited argumentation allows agents to provide a reason for rejecting an offer and guide future offers to an acceptable agreement. The protocol also supports symmetry of roles. The protocol affords provider and consumer roles with the same abilities, permissions and (data) access. Concurrent negotiation sessions enable agents to negotiate with multiple counter-agents simultaneously.

Additional mechanisms and design consideration ensure reliability and security of the negotiation protocol. First, the protocol is decentralized. Provider and consumer agents communicate directly with one another. The negotiation protocol uses no centralized negotiation component, such as a broker. Therefore, failure of any single component (e.g. a single crashed agent) does not affect the other ongoing negotiation sessions. The underlying middleware (e.g. AgentScape) provides additional mechanisms for fault tolerance, data persistence and recovery. Confidentiality and integrity of negotiation messages and (sensitive) data is assured with additional monitoring mechanism.

`RQ3` Can agreements be enforced in a transparent and trustworthy manner?

`RQ4` Can trust be established and maintained between agents in untrusted environments?

Chapter 4 presents a self-adaptive monitoring approach that detects agreement violations and penalizes offending agents. An impartial party (e.g. TTP) performs and stores monitoring measurements. A TTP guarantees that all agents are treated equally and no role (e.g. provider) receives special privileges, such as special access to monitoring data. Monitoring is therefore transparent to all participants and monitoring results are trustworthy.

Open environments lack a single authority to govern all agents and punish malicious activity. Therefore, consumer and provider agents face a risk of entering negotiation or exchanging services with potentially malicious agents. The self-adaptive monitoring approach reacts to changes in levels of perceived risk by increasing or decreasing the level of monitoring (i.e. passive or active mode, interval length). Perceived levels of risk consider reputation (i.e. history of transactions) of a given agent. Agreement compliance increases reputation and builds trust between agents. The adaptive monitor responds to increased levels of trust by reducing assurance and associated costs. Thus, the monitor rewards agents for begin trustworthy.

The general research question of this dissertation asks:

Can a Multi Agent System (MAS) frameworks be designed to support automated negotiation and monitoring of services in dynamic, distributed, open environments?

This dissertation presents a negotiation protocol for MAS (Chapter 3) and a self-adaptive monitoring approach (Chapter 4) for open environments. Chapter 5 combines and applies this protocol and approach to example open environments. Use cases in the Smart Energy Grid and Cloud environments demonstrate the practical application of the contributions of this dissertation.

6.2 Future Work

This section provides an overview of possible future work building on the research presented in this dissertation. This section broadly organizes future work under negotiation research and monitoring research.

6.2.1 Future Negotiation Research

Creation of complex SLAs remains a challenge. For instance, a consumer requires storage and compute power. One CSP offers the lowest price for storage, while another the lowest price for compute power. This situation currently requires at least two negotiation sessions and two, separate SLAs. Research on compositional SLAs provides examples of incorporating two, separate SLAs into a single SLA [43].

Expressing dynamic relationships and preferences in SLAs remains a challenge. For instance, as a deadline approaches, the need for a successful agreement increases and the need for a particular attribute decreases. Future research may also focus on the challenge of adapting negotiation strategies and utility functions during dynamic negotiations.

6.2.2 Future Monitoring Research

Future work may investigate additional trust mechanisms, such as reputation authorities [79]. Incorporating separate reputation authorities will aid the problem faced when two agents negotiate for the first time and thus have no previous experience or history of transactions. When negotiating with a previously unknown agent, a reputation authority provides an estimation of that agent's reputation.

While the results of experimentation with the monitoring framework has indicated the suitability of a decentralized approach for monitoring SLAs under heavy load, further experimentation may investigate more security and reliability issues, such as active attacks against monitors. While the current implementation provides a working basis, more work may focus on standardization of penalties and proof of violation.

The incorporation of secure hardware modules (e.g. Trusted Computing Platform [127]), to support monitoring sensors, may be researched.

The feasibility of the self-adaptive monitoring approach in a real-world setting partially depends on the cost structure of a CSP. Monitoring SLAs with an external TTP has associated costs and performance overhead. Likewise, implementing a self-adaptive monitoring framework also has associated costs, but also offers benefits to both a CSP and their consumers. A CSP must analyze these costs to determine if a self-adaptive monitoring framework, and the benefits it offers, are economically appealing. Economic considerations are left to future work.

The simulation of the energy marketplace may be improved with real-world usage/consumption statistics and realistic economic models.

This dissertation incorporates transparency and symmetry (i.e. equality) into the design of automated negotiation and monitoring systems to promote trust. Future qualitative studies can measure the impact of these design choices on social aspects, including usability and user acceptance of automated negotiation systems.

6.3 Conclusions

Automation of complex tasks, such as negotiation, reduces manual burden, increases efficiency and reduces (human) errors. Software agents encapsulate human characteristics, such as coordination and adaptation. Agent technology provides an intuitive interface for users to access the potential of automation. This dissertation designs a Multi Agent System (MAS) framework to support automated negotiation in dynamic, open environments, such as the Smart Energy Grid and the Cloud.

In the case of the Smart Energy Grid, the looming complexity crisis of intermittent generation, real-time pricing and consumer demand shifting requires immediate attention. This domain presents not only technical (e.g. smart-meters) but also social challenges (e.g. user acceptance). This dissertation presents a MAS automation framework that addresses technical challenges by reducing manual labor and increasing efficiency. Automation even enables higher utilization of green resources and reduction of waste (e.g. produced,

but unconsumed energy). Transparent, trusted monitoring mechanisms address social challenges by ensuring privacy of (sensitive) data and encouraging user acceptance.

In the case of the Cloud, the model of elastic computing enables consumers to access (and pay for) the exact amount of computing power required. Cloud adoption promises many benefits, including reduced costs and environmental pollution[2]. However, vendor lock-in prevents consumers from accessing the full potential of Cloud services. Standardization of Cloud formats (e.g. platform/API, hypervisor, disk images) allows free migration of services between CSPs. The MAS solution presented in this dissertation assists consumers discovering, choosing, negotiating, migrating and monitoring Cloud services. Seamless, automated migration of data and services creates new possibilities. For instance, data can follow cheap energy supply by migrating to follow nighttime tariffs.

Other domains also benefit from the automated negotiation technologies proposed in this dissertation. For instance, agents may reduce vehicle traffic by negotiating with agents representing other vehicles and choosing the optimal route to destination [55]. Agents may also negotiate with airlines to lower ticket cost [172]. The future brings opportunity.

[2]Green-powered data centers (e.g. GreenQloud) perform CPU intensive data processing rather than in-house servers, ultimately powered by coal.

Bibliography

[1] Aknine, S., Pinson, S., and Shakun, M. F. An extended multi-agent negotiation protocol. *Autonomous Agents and Multi-Agent Systems 8*, 1 (2004), 5–45.

[2] Alhamad, M., Dillon, T., and Chang, E. Conceptual sla framework for cloud computing. In *Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on* (2010), IEEE, pp. 606–610.

[3] Allcott, H. Rethinking real-time electricity pricing. *Resource and Energy Economics 33*, 4 (2011), 820–842.

[4] Anandasivam, A., and Premm, M. Bid Price Control and Dynamic Pricing in Clouds. In *17th European Conference on Information Systems (ECIS 2009), Verona, Italy* (2009), pp. 328–341.

[5] Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., and Xu, M. Web Services Agreement Specification (WS-Agreement) GFD-R.192. Tech. rep., Global Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG, 2011.

[6] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. A view of cloud computing. *Communications of the ACM 53*, 4 (2010), 50–58.

[7] Atkinson, K., Bench-Capon, T., and Mcburney, P. A dialogue game protocol for multi-agent argument over proposals for action. *Autonomous Agents and Multi-Agent Systems 11*, 2 (2005), 153–171.

[8] Bal, H., Bhoedjang, R., Hofman, R., Jacobs, C., Kielmann, T., Maassen, J., Van Nieuwpoort, R., Romein, J., Renambot, L., Rühl, T., et al. The distributed asci supercomputer project. *ACM SIGOPS Operating Systems Review 34*, 4 (2000), 76–96.

[9] Baliga, J., Ayre, R., Hinton, K., and Tucker, R. Green cloud computing: Balancing energy in processing, storage, and transport. *Proceedings of the IEEE 99*, 1 (2011), 149–167.

[10] Barbose, G., Goldman, C., and Neenan, B. A survey of utility experience with real time pricing. Tech. rep., Ernest Orlando Lawrence Berkeley National Laboratory, Berkeley, CA (US), 2004.

[11] Bartolini, C., Preist, C., and Jennings, N. A software framework for automated negotiation. In *Software Engineering for Multi-Agent Systems III*, R. Choren, A. Garcia, C. Lucena, and A. Romanovsky, Eds., vol. 3390 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2005, pp. 213–235.

[12] Battre, D., Brazier, F., Clark, K., Oey, M., Papaspyrou, A., Wäldrich, O., Wieder, P., and Ziegler, W. A proposal for ws-agreement negotiation. In *11th IEEE/ACM International Conference on Grid Computing* (2010).

[13] Beam, C., and Segev, A. Automated negotiations: A survey of the state of the art. *Wirtschaftsinformatik 39*, 3 (1997), 263–268.

[14] Binmore, K. *Game theory: a very short introduction*. Oxford University Press, USA, 2007.

[15] Boneh, D., Lynn, B., and Shacham, H. Short signatures from the Weil pairing. *Journal of Cryptology 17*, 4 (2004), 297–319.

[16] Borenstein, S. The trouble with electricity markets: understanding california's restructuring disaster. *The Journal of Economic Perspectives 16*, 1 (2002), 191–211.

[17] Borenstein, S. The long-run efficiency of real-time electricity pricing. *Energy Journal 26*, 3 (2005), 93–116.

[18] Boyle, J., Cohen, R., Herzog, S., Rajan, R., and Sastry, A. The cops (common open policy service) protocol. Tech. rep., RFC Editor, 2000.

[19] Brazier, F., Cornelissen, F., Gustavsson, R., Jonker, C., Lindeberg, O., Polak, B., and Treur, J. A multi-agent system performing one-to-many negotiation for load balancing of electricity use. *Electronic Commerce Research and Applications 1*, 2 (2002), 208–224.

[20] Brazier, F., Kubbe, O., Oskamp, A., Wijngaards, N., et al. Are law-abiding agents realistic. In *Proceedings of the workshop on the Law of Electronic Agents (LEA02)* (2002), pp. 151–155.

[21] Brazier, F., Ogston, E., and Warnier, M. The future of energy markets and the challenge of decentralized self-management. In *Agents and Peer-to-Peer Computing*. Springer, 2012, pp. 95–103.

[22] Brazier, F. M., Kephart, J. O., Van Dyke Parunak, H., and Huhns, M. N. Agents and service-oriented computing for autonomic computing: A research agenda. *Internet Computing, IEEE 13*, 3 (2009), 82–87.

[23] Buyya, R., and Vazhkudai, S. Compute power market: Towards a market-oriented grid. In *ccgrid* (2001), Published by the IEEE Computer Society, p. 574.

[24] Buyya, R., Yeo, C., and Venugopal, S. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on* (2008), Ieee, pp. 5–13.

[25] Chao, K., Anane, R., Chen, J., and Gatward, R. Negotiating agents in a market-oriented grid. In *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on* (may 2002), p. 436.

[26] Cheng, S., Chang, C., Zhang, L., and Kim, T. Towards competitive web service market. In *Future Trends of Distributed Computing Systems, 2007. FTDCS'07. 11th IEEE International Workshop on* (2007), IEEE, pp. 213–219.

[27] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2* (2005), NSDI'05, USENIX Association, pp. 273–286.

[28] Clark, K., van Splunter, S., Warnier, M., and Brazier, F. Expressing intervals in automated service negotiation. In *Grids and Service-Oriented Architectures for Service Level Agreements*, P. Wieder, R. Yahyapour, and W. Ziegler, Eds., CoreGRID. Springer-Verlag, New York, NY, USA, 2010, pp. 67–76.

[29] Clark, K., Warnier, M., and Brazier, F. Self-adaptive service monitoring. In *proceedings of the 2011 International Conference on Adaptive and Intelligent Systems (ICAIS 2011)* (2011), Springer, pp. 119–130.

[30] Clark, K., Warnier, M., and Brazier, F. An intelligent cloud resource allocation service - agent-based automated cloud resource allocation using micro-agreements. In *In the proceedings of the 2nd International Conference on Cloud Computing and Services Science (CLOSER 2012)* (2012), pp. 37–45.

[31] Clark, K., Warnier, M., and Brazier, F. Automated non-repudiable cloud resource allocation. In *Cloud Computing and Services Science*. Springer, 2013, pp. 168–182.

[32] Clark, K., Warnier, M., and Brazier, F. Increasing green energy market efficiency using micro agreements. In *Green ICT & Energy: From Smart to Wise Strategies*, J. H. Appelman, A. Osseyran, and M. Warnier, Eds., Sustainable Energy Developments. CRC Press, 2013, pp. 77–91.

[33] Clark, K., Warnier, M., Quillinan, T., and Brazier, F. Secure monitoring of service level agreements. In *Fifth International Conference on Availability, Reliability and Security (ARES 2010)* (March 2010), IEEE, pp. 454–461.

[34] Clark, K. P., Warnier, M., and Brazier, F. Self-Adaptive Service Level Agreement Monitoring in Cloud Environments. *Multiagent and Grid Systems 9* (2013).

[35] Clayman, S., Galis, A., Chapman, C., Toffetti, G., Rodero-Merino, L., Vaquero, L., Nagin, K., and Rochwerger, B. Monitoring service clouds in the future internet. In *Towards the Future Internet-Emerging Trends from European Research*. IOS Press, 2010, pp. 115–126.

[36] Commission, E., et al. Investing in the development of low carbon technologies (set-plan). *COM (2009) 519* (2009).

[37] Comuzzi, M., Kotsokalis, C., Spanoudakis, G., and Yahyapour, R. Establishing and monitoring slas in complex service based systems. In *Web Services, 2009. ICWS 2009. IEEE International Conference on* (2009), Ieee, pp. 783–790.

[38] Connolly, D., Lund, H., Mathiesen, B. V., Pican, E., and Leahy, M. The technical and economic implications of integrating fluctuating renewable energy using energy storage. *Renewable Energy 43* (2012), 47–60.

[39] Coulouris, G., J., D., and Kindberg, T. *Distributed Systems: Concepts and Design*, vol. Fifth Edition. Addison-Wesley, 2012.

[40] Cramton, P. Electricity market design: The good, the bad, and the ugly. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on* (2003), IEEE, pp. 8–pp.

[41] Creswell, J. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*, 2nd ed. SAGE Publications, 2003.

[42] Crosby, S., Doyle, R., Gering, M., Gionfriddo, M., et al. Open virtualization format specification 1.1.0. Tech. rep., DSP0243, Distributed Management Task Force, Inc, 2010.

[43] Czajkowski, K., Foster, I., Kesselman, C., Sander, V., and Tuecke, S. Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *Job scheduling strategies for parallel processing* (2002), Springer, pp. 153–183.

[44] De Bruijne, M., and Van Eeten, M. Systems that should have failed: critical infrastructure protection in an institutionally fragmented environment. *Journal of Contingencies and Crisis Management 15*, 1 (2007), 18–29.

[45] De Paepe, M., D'Herdt, P., and Mertens, D. Micro-chp systems for residential applications. *Energy conversion and management 47*, 18 (2006), 3435–3446.

[46] Deindl, M., Block, C., Vahidov, R., and Neumann, D. Load shifting agents for automated demand side management in micro energy grids. In *Self-Adaptive and Self-Organizing Systems, 2008. SASO '08. Second IEEE International Conference on* (oct. 2008), pp. 487 –488.

[47] Dielmann, K., and van der Velden, A. Virtual power plants (vpp)-a new perspective for energy generation? In *Modern Techniques and Technologies, 2003. MTT 2003. Proceedings of the 9th International Scientific and Practical*

Conference of Students, Post-graduates and Young Scientists (2003), IEEE, pp. 18–20.

[48] Dikaiakos, M. D., and Zeinalipour-Yazti, D. A distributed middleware infrastructure for personalized services. *Computer Communications 27*, 15 (2004), 1464–1480.

[49] Dimeas, A., and Hatziargyriou, N. Control agents for real microgrids. In *Intelligent System Applications to Power Systems, 2009. ISAP'09. 15th International Conference on* (2009), IEEE, pp. 1–5.

[50] Dobson, G., and Sanchez-Macian, A. Towards unified QoS/SLA ontologies. *IEEE Services Computing Workshops, 2006. SCW'06* (2006), 169–174.

[51] Duan, R., and Deconinck, G. Future electricity market interoperability of a multi-agent model of the Smart Grid. In *Networking, Sensing and Control (ICNSC), 2010 International Conference on* (2010), IEEE, pp. 625–630.

[52] Ermolayev, V., and Keberle, N. A generic ontology of rational negotiation. In *Information Systems Technology and its Applications. 5th Int. Conf. ISTA* (2006), pp. 30–31.

[53] Fabrice, B. Qemu, a fast and portable dynamic translator. In *USENIX 2005 Annual Technical Conference, FREENIX Track* (2005), pp. 41–46.

[54] Ferretti, S., Ghini, V., Panzieri, F., Pellegrini, M., and Turrini, E. Qos-aware clouds. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on* (july 2010), pp. 321 –328.

[55] Fiosins, M., Fiosina, J., Müller, J. P., and Görmer, J. Agent-based integrated decision making for autonomous vehicles in urban traffic. In *Advances on Practical Applications of Agents and Multiagent Systems*. Springer, 2011, pp. 173–178.

[56] FIPA. Fipa acl message structure specification. Tech. rep., Foundation for Intelligent Physical Agents, 2002.

[57] FIPA. Fipa iterated contract net interaction protocol specification. Tech. rep., Foundation for Intelligent Physical Agents, 2002.

[58] Foster, I. Globus Toolkit Version 4: Software for Service-Oriented Systems. In *International Conference on Network and Parallel Computing (IFIP)* (2006), vol. 3379 of *LNCS*, Springer-Verlag, pp. 2–13.

[59] Foster, I., Zhao, Y., Raicu, I., and Lu, S. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08* (2008), IEEE, pp. 1–10.

[60] Ganek, A., and Corbi, T. The dawning of the autonomic computing era. *IBM Systems Journal 42*, 1 (2003), 5–18.

[61] Gellings, C. *The smart grid: enabling energy efficiency and demand response.* CRC, 2009.

[62] Gottwalt, S., Ketter, W., Block, C., Collins, J., and Weinhardt, C. Demand side management—a simulation of household behavior under variable prices. *Energy Policy* (2011).

[63] Green, L., Mirchandani, V., Cergol, I., and Verchere, D. Design of a dynamic sla negotiation protocol for grids. In *Proceedings of the first international conference on Networks for grid applications* (2007), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), p. 13.

[64] Gudgin, M., Hadley, M., and Rogers, T. Web services addressing 1.0-core. Tech. rep., W3C Recommendation, May 2006.

[65] Gymnopoulos, L., Dritsas, S., Gritzalis, S., and Lambrinoudakis, C. GRID security review. In *Lecture Notes in Computer Science*. Springer, 2003, pp. 100–111.

[66] Hatziargyriou, N., Asano, H., Iravani, R., and Marnay, C. Microgrids. *Power and Energy Magazine, IEEE 5*, 4 (july-aug. 2007), 78 –94.

[67] Hayashibara, N., Cherif, A., and Katayama, T. Failure detectors for large-scale distributed systems. In *Reliable Distributed Systems, 2002. Proceedings. 21st IEEE Symposium on* (2002), IEEE, pp. 404–409.

[68] Hevner, A. R., March, S. T., Park, J., and Ram, S. Design science in information systems research. *MIS Q. 28*, 1 (Mar. 2004), 75–105.

[69] Hindriks, K., Jonker, C., and Tykhonov, D. Negotiation Dynamics: Analysis, Concession Tactics, and Outcomes. In *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology* (2007), IEEE Computer Society Washington, DC, USA, pp. 427–433.

[70] Hirschheim, R. Information systems epistemology: An historical perspective. In *Information Systems Research: Issues, Methods and Practical Guidelines*. Blackwell Scientific Publications, London, U.K., 1992, pp. 28–60.

[71] Honing, N., and La Poutré, H. Reduction of market power and stabilisation of outcomes in a novel and simplified two-settlement electricity market. In *Proceedings of the The 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology - Volume 02* (Washington, DC, USA, 2012), WI-IAT '12, IEEE Computer Society, pp. 103–110.

[72] Hsu, C. Dominant factors for online trust. In *Cyberworlds, 2008 International Conference on* (September 2008), pp. 165 –172.

[73] Huang, H., Zhu, G., and Jin, S. Revisiting Trust and Reputation in Multi-agent Systems. In *Computing, Communication, Control, and Management, 2008. CCCM'08. ISECS International Colloquium on* (2008), vol. 1.

[74] Jennings, N. An agent-based approach for building complex software systems. *Communications of the ACM 44*, 4 (2001), 35–41.

[75] Jennings, N., Faratin, P., Lomuscio, A., Parsons, S., Wooldridge, M., and Sierra, C. Automated negotiation: prospects, methods and challenges. *Group Decision and Negotiation 10*, 2 (2001), 199–215.

[76] Jennings, N., and Wooldridge, M., Eds. *Applications of Intelligent Agents*. Agent Technology: Foundations, Applications, and Markets. Springer, 1998, ch. 1, pp. 3–28.

[77] Jin, H., and Wu, H. Semantic-enabled specification for Web Services agreement. *International Journal of Web Services Practices 1*, 1-2 (2005), 13–20.

[78] Jonker, C., and Treur, J. An agent architecture for multi-attribute negotiation. In *International joint conference on artificial intelligence* (2001), vol. 17, LAWRENCE ERLBAUM ASSOCIATES LTD, pp. 1195–1201.

[79] Jøsang, A., Ismail, R., and Boyd, C. A survey of trust and reputation systems for online service provision. *Decision Support Systems 43*, 2 (2007), 618–644.

[80] Katsaros, G., Kousiouris, G., Gogouvitis, S. V., Kyriazis, D., Menychtas, A., and Varvarigou, T. A self-adaptive hierarchical monitoring mechanism for clouds. *Journal of Systems and Software 85*, 5 (2012), 1029–1041.

[81] Keller, A., and Ludwig, H. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management 11*, 1 (2003), 57–81.

[82] Kephart, J., and Chess, D. The vision of autonomic computing. *Computer 36*, 1 (2003), 41–50.

[83] Keung, H., Dyson, J., Jarvis, S., and Nudd, G. Self-adaptive and self-optimising resource monitoring for dynamic grid environments. In *Database and Expert Systems Applications, 15th International Workshop on* (August 2004), pp. 689 – 693.

[84] Khader, D., Padget, J., and Warnier, M. Reactive monitoring of service level agreements. In *Grids and Service-Oriented Architectures for Service Level Agreements*, P. Wieder, R. Yahyapour, and W. Ziegler, Eds., CoreGRID. Springer, 2010, pp. 13–22.

[85] Klos, T., Somefun, K., and La Poutré, H. Automated interactive sales processes. *IEEE Intelligent Systems 26*, 4 (2011), 54–61.

[86] Koritarov, V. Real-world market representation with agents. *Power and Energy Magazine, IEEE 2*, 4 (2004), 39–46.

[87] Kraus, S. Automated negotiation and decision making in multiagent environments. In *Multi-Agent Systems and Applications*, M. Luck, V. Mařík, O. Štěpánková, and R. Trappl, Eds., vol. 2086 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2001, pp. 150–172.

[88] Krummenacher, R., Blunder, D., Simperl, E., and Fried, M. An open distributed middleware for the semantic web. In *International Conference on Semantic Systems (I-SEMANTICS)* (2009).

[89] Latham, D. Department of Defense Trusted Computer System Evaluation Criteria. *Department of Defense* (1986).

[90] Lau, F., Rubin, S., Smith, M., and Trajkovic, L. Distributed denial of service attacks. In *2000 IEEE International Conference on Systems, Man, and Cybernetics* (2000), vol. 3, pp. 2275–2280.

[91] Lazarewicz, M., and Rojas, A. Grid frequency regulation by recycling electrical energy in flywheels. In *Power Engineering Society General Meeting, 2004. IEEE* (june 2004), pp. 2038 –2042 Vol.2.

[92] Lin, R., Kraus, S., Baarslag, T., Tykhonov, D., Hindriks, K., and Jonker, C. Genius: An integrated environment for supporting the design of generic automated negotiators. *Computational Intelligence* (2012).

[93] Lin, R., Kraus, S., Wilkenfeld, J., and Barry, J. Negotiating with bounded rational agents in environments with incomplete information using an automated agent. *Artificial Intelligence 172*, 6 (2008), 823–851.

[94] Ludwig, A., Braun, P., Kowalczyk, R., and Franczyk, B. A Framework for Automated Negotiation of Service Level Agreements in Services Grids. In *Business Process Management Workshops* (2006), vol. 3812 of *Lecture Notes in Computer Science*, Springer, pp. 89–101.

[95] Ludwig, H., Dan, A., and Kearney, R. Cremona: an architecture and library for creation and monitoring of WS-agreents. In *Proceedings of the 2nd international conference on Service oriented computing* (2004), ACM New York, NY, USA, pp. 65–74.

[96] Ludwig, H., Keller, A., Dan, A., King, R., and Franck, R. Web Service Level Agreement (WSLA) Language Specification. Tech. rep., IBM Corporation, 2003.

[97] Ludwig, H., Nakata, T., Wäldrich, O., Wieder, P., and Ziegler, W. Reliable orchestration of resources using ws-agreement. *Lecture Notes in Computer Science 4208* (2006), 753.

[98] Luttikhuis, P. Duitsland verkijkt zich op stroom uit zon en wind. *NRC Handelsblad* (June 2012).

[99] Mach, R., Lepro-Metz, R., Jackson, S., and McGinnis, L. Usage Record Format Recommendation (GFD-RP 098). In *Open Grid Forum Recommendation* (2007).

[100] Mach, W., and Schikuta, E. A generic negotiation and re-negotiation framework for consumer-provider contracting of web services. In *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services* (2012), ACM, pp. 348–351.

[101] Manchala, D. Trust metrics, models and protocols for electronic commerce transactions. In *Distributed Computing Systems, 1998. Proceedings. 18th International Conference on* (1998), pp. 312–321.

[102] Manchala, D. E-commerce trust metrics and models. *Internet Computing, IEEE 4*, 2 (March 2000), 36 –44.

[103] Massie, M., Chun, B., and Culler, D. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing 30*, 7 (2004), 817–840.

[104] McCrone, A., Usher, E., Sonntag-O'Brien, V., Moslener, U., Andreas, J., and Gruening, C. Global trends in renewable energy investment 2011. Tech. rep., United Nations Environment Programme (UNEP) and Bloomberg New Energy Finance, 2011.

[105] McCumber, J. Information systems security: A comprehensive model. In *Proceedings of the 14th NIST National Computer Security Conference* (Washington, D.C., oct. 1991), pp. 328–337.

[106] Mirkovic, J., and Reiher, P. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review 34*, 2 (2004), 39–53.

[107] Mobach, D. *Agent-Based Mediated Service Negotiation*. PhD thesis, Computer Science Department, Vrije Universiteit Amsterdam, May 2007.

[108] Mobach, D., Overeinder, B., and Brazier, F. A WS-Agreement Based Resource Negotiation Framework for Mobile Agents. *Scalable Computing: Practice and Experience 7*, 1 (2006), 23–36.

[109] Mobach, D., Overeinder, B., Brazier, F., and Dignum, F. A two-tiered model of negotiation based on web service agreements. In *Proceedings of the Third European Workshop on Multi-Agent Systems (EUMAS'05)* (December 2005), pp. 202–213.

[110] Munawar, M., and Ward, P. Adaptive monitoring in enterprise software systems. In *First Workshop on Tackling Computer Systems Problems with Machine Learning (SysML)* (June 2006).

[111] Nadiminti, K., De Assunção, M. D., and Buyya, R. Distributed systems and recent innovations: Challenges and benefits. *InfoNet Magazine 16*, 3 (2006), 1–5.

[112] NagiosEnterprises. Nagios - the industry standard in it infrastructure monitoring. `http://nagios.org/`, May 2011.

[113] Netzer, R. H. B., and Miller, B. P. What are race conditions?: Some issues and formalizations. *ACM Lett. Program. Lang. Syst. 1*, 1 (Mar. 1992), 74–88.

[114] Nguyen, T. M. T., Boukhatem, N., Doudane, Y. G., and Pujolle, G. Cops-sls: a service level negotiation protocol for the internet. *Communications Magazine, IEEE 40*, 5 (2002), 158–165.

[115] Niehörster, O., Brinkmann, A., Fels, G., Kruger, J., and Simon, J. Enforcing slas in scientific clouds. In *Cluster Computing (CLUSTER), 2010 IEEE International Conference on* (2010), IEEE, pp. 178–187.

[116] NL, A. Zonnestroom en de nederlandse wetgeving (dutch). Ministerie van Economische Zaken, Landbouw en Innovatie, https://www.agentschapnl.nl/content/zonnestroom-en-de-nederlandse-wetgeving, 2012.

[117] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. The eucalyptus open-source cloud-computing system. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on* (2009), IEEE, pp. 124–131.

[118] Ogston, E., and Brazier, F. Apportionment of control in virtual power stations. In *In the proceedings of the international conference on infrastructure systems and services 2009: Developing 21st Century Infrastructure Networks* (2009).

[119] OpenStack. Openstack: Open source software for building private and public clouds. `http://www.openstack.org`, 2011.

[120] Ouelhadj, D., Garibaldi, J., MacLaren, J., Sakellariou, R., and Krishnakumar, K. A Multi-agent Infrastructure and a Service Level Agreement Negotiation Protocol for Robust Scheduling in Grid Computing. In *Advances in Grid Computing-EGC 2005* (2005), Springer, pp. 651–660.

[121] Overeinder, B., and Brazier, F. Scalable Middleware Environment for Agent-Based Internet Applications. *Applied Parallel Computing. State of the Art in Scientific Computing 3732* (2005), 675–679.

[122] Overeinder, B. J., Posthumus, E., and Brazier, F. Integrating peer-to-peer networking and computing in the agentscape framework. In *Peer-to-Peer Computing, 2002.(P2P 2002). Proceedings. Second International Conference on* (2002), IEEE, pp. 96–103.

[123] Padgett, J., Djemame, K., and Dew, P. Grid-Based SLA Management. In *Advances in Grid Computing (EGC 2005)*. Springer, 2005, pp. 1076–1085.

[124] Palensky, P., and Dietrich, D. Demand side management: Demand response, intelligent energy systems, and smart loads. *Industrial Informatics, IEEE Transactions on 7*, 3 (2011), 381–388.

[125] Pallickara, S., and Fox, G. Naradabrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids. In *Middleware 2003* (2003), Springer, pp. 41–61.

[126] Parashar, M., and Hariri, S. Autonomic computing: An overview. In *Unconventional Programming Paradigms*. Springer, 2005, pp. 257–269.

[127] Pearson, S. Trusted computing platforms, the next security solution. Tech. rep., HP Labs, 2002.

[128] Peterson, S. B., Whitacre, J., and Apt, J. The economics of using plug-in hybrid electric vehicle battery packs for grid storage. *Journal of Power Sources 195*, 8 (2010), 2377 – 2384.

[129] Piwko, R., Osborn, D., Gramlich, R., Jordan, G., Hawkins, D., and Porter, K. Wind energy delivery issues [transmission planning and competitive electricity market operation]. *Power and Energy Magazine, IEEE 3*, 6 (2005), 47–56.

[130] Pueschel, T., Anandasivam, A., Buschek, S., and Neumann, D. Making Money With Clouds: Revenue Optimization Through Automated Policy Decisions. In *17th European Conference on Information Systems (ECIS 2009), Verona, Italy* (2009), pp. 355–367.

[131] Quillinan, T., Brazier, F., Aldewereld, H., Dignum, F., Dignum, V., Penserini, L., and Wijngaards, N. Developing agent-based organizational models for crisis management. In *Proc. of the 8th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS 2009)* (2009), pp. 45–51.

[132] Quillinan, T., Clark, K., Warnier, M., Brazier, F., and Rana, O. Negotiation and monitoring of service level agreements. In *Grids and Service-Oriented Architectures for Service Level Agreements*, P. Wieder, R. Yahyapour, and W. Ziegler, Eds., CoreGRID. Springer-Verlag, New York, NY, USA, 2010, pp. 167–176.

[133] Quillinan, T., Clayton, B., and Foley, S. GridAdmin: Decentralising grid administration using trust management. In *Parallel and Distributed Computing, 2004. Third International Symposium on/Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, 2004. Third International Workshop on* (2004), IEEE, pp. 184–192.

[134] Quillinan, T., Warnier, M., Oey, M., Timmer, R., and Brazier, F. Enforcing Security in the AgentScape Middleware. In *Proceedings of the 1st International Workshop on Middleware Security (MidSec)* (December 2008), ACM.

[135] Quinn, M. *Parallel computing: theory and practice*. McGraw-Hill, Inc., 1994.

[136] Rahwan, I., Kowalczyk, R., and Pham, H. H. Intelligent agents for automated one-to-many e-commerce negotiation. *Aust. Comput. Sci. Commun. 24*, 1 (Jan. 2002), 197–204.

[137] Raiffa, H. *The art and science of negotiation: How to resolve conflicts and get the best out of bargaining*. Belknap Press, 2002.

[138] Ramchurn, S., Vytelingum, P., Rogers, A., and Jennings, N. Putting the 'smarts' into the smart grid: A grand challenge for artificial intelligence. *Communications of the ACM* (2012).

[139] Ramezani, S., Bosman, P., and La Poutré, H. Adaptive strategies for dynamic pricing agents. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM International Conference on* (2011), vol. 2, pp. 323–328.

[140] Rana, O., Warnier, M., Quillinan, T., and Brazier, F. Monitoring and Reputation Mechanisms for Service Level Agreements. In *Proceedings of the 5th International Workshop on Grid Economics and Business Models (GenCon)* (Las Palmas, Gran Canaria, Spain., August 2008), Springer Verlag.

[141] Rana, O., Warnier, M., Quillinan, T., Brazier, F., and Cojocarasu, D. Managing violations in service level agreements. In *Grid Middleware and Services*. Springer, 2008, pp. 349–358.

[142] Reed, C., and Walton, D. Towards a formal and implemented model of argumentation schemes in agent communication. *Autonomous Agents and Multi-Agent Systems 11*, 2 (2005), 173–188.

[143] Resnick, P., Kuwabara, K., Zeckhauser, R., and Friedman, E. Reputation systems. *Communications of the ACM 43*, 12 (2000), 45–48.

[144] Rivest, R., Shamir, A., and Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21*, 2 (1978), 120–126.

[145] Romano, L., De Mari, D., Jerzak, Z., and Fetzer, C. A novel approach to qos monitoring in the cloud. In *Data Compression, Communications and Processing (CCP), 2011 First International Conference on* (june 2011), pp. 45 –51.

[146] Romp, G. *Game theory: introduction and applications*. Oxford university press, 1997.

[147] Rosenschein, J. S., and Zlotkin, G. Designing conventions for automated negotiation. *AI magazine 15*, 3 (1994), 29.

[148] Sahai, A., Durante, A., and Machiraju, V. Towards automated sla management for web services. Tech. Rep. Research Report HPL-2001-310 (R. 1), Hewlett-Packard, 2002.

[149] Sahai, A., Graupner, S., Machiraju, V., and van Moorsel, A. Specifying and monitoring guarantees in commercial grids through SLA. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on* (2003), pp. 292–299.

[150] Sahai, A., Machiraju, V., Sayal, M., Van Moorsel, A., and Casati, F. Automated sla monitoring for web services. In *Management Technologies for E-Commerce and E-Business Applications*. Springer, 2002, pp. 28–41.

[151] Sandholm, T. An implementation of the contract net protocol based on marginal cost calculations. In *AAAI* (1993), vol. 93, pp. 256–262.

[152] Sandholm, T. Agents in electronic commerce: Component technologies for automated negotiation and coalition formation. *Autonomous Agents and Multi-Agent Systems 3*, 1 (2000), 73–96.

[153] Sandholm, T. W., and Lesser, V. R. Issues in automated negotiation and electronic commerce: Extending the contract net framework. In *Proceedings of the 1st International Conference on Multiagent Systems* (1995), pp. 328–335.

[154] Schneier, B. *Applied Cryptography: protocols, algorithms, and source code in C*, 2nd ed. John Wiley and Sons, 1996.

[155] Schroeder, M. An efficient argumentation framework for negotiating autonomous agents. In *Multi-Agent System Engineering*. Springer, 1999, pp. 140–149.

[156] Shvaiko, P., and Euzenat, J. Ten challenges for ontology matching. In *On the Move to Meaningful Internet Systems: OTM 2008*. Springer, 2008, pp. 1164–1182.

[157] Shvaiko, P., and Euzenat, J. Ontology matching: state of the art and future challenges. *Knowledge and Data Engineering, IEEE Transactions on 25*, 1 (2013), 158–176.

[158] Sjoberg, D. I., Dyba, T., and Jorgensen, M. The future of empirical methods in software engineering research. In *Future of Software Engineering, 2007. FOSE'07* (2007), IEEE, pp. 358–378.

[159] Smith, R. The contract net protocol: High-level communication and control in a distributed problem solver. *Computers, IEEE Transactions on 100*, 12 (1980), 1104–1113.

[160] Stantchev, V., and Schröpfer, C. Negotiating and enforcing QoS and SLAs in grid and cloud computing. In *Advances in Grid and Pervasive Computing*. Springer, 2009, pp. 25–35.

[161] Strbac, G. Demand side management: Benefits and challenges. *Energy Policy 36*, 12 (2008), 4419–4426.

[162] Ströbel, M., and Weinhardt, C. The montreal taxonomy for electronic negotiations. *Group Decision and Negotiation 12*, 2 (2003), 143–164.

[163] Tajeddine, A., Kayssi, A., Chehab, A., and Artail, H. A comprehensive reputation-based trust model for distributed systems. In *Security and Privacy for Emerging Areas in Communication Networks, 2005. Workshop of the 1st International Conference on* (September 2005), pp. 116 – 125.

[164] Tamma, V., Phelps, S., Dickinson, I., and Wooldridge, M. Ontologies for supporting negotiation in e-commerce. *Engineering applications of artificial intelligence 18*, 2 (2005), 223–236.

[165] Tanenbaum, A. S., and Van Steen, M. *Distributed systems*, vol. 2. Prentice Hall, 2002.

[166] Tondello, G., and Siqueira, F. The QoS-MO ontology for semantic QoS modeling. In *Proceedings of the 2008 ACM symposium on Applied computing* (2008), ACM New York, NY, USA, pp. 2336–2340.

[167] Van het Schip, R., van Splunter, S., and Brazier, F. Template evaluation and selection for ws-agreement. In *Service Level Agreements in Grids Workshop proceedings* (2009).

[168] Van Steen, M., Pierre, G., and Voulgaris, S. Challenges in very large distributed systems. *Journal of Internet Services and Applications 3*, 1 (2012), 59–66.

[169] Van Veenen, J., and Prakken, H. A protocol for arguing about rejections in negotiation. In *Argumentation in Multi-Agent Systems*. Springer, 2006, pp. 138–153.

[170] Venables, M. Smart meters make smart consumers. *Engineering Technology 2*, 4 (april 2007), 23.

[171] Vokřínek, J., Bíba, J., Hodík, J., Vybíhal, J., and Pěchouček, M. Competitive contract net protocol. In *SOFSEM 2007: Theory and Practice of Computer Science*. Springer, 2007, pp. 656–668.

[172] Vukmirovic, M., Ganzha, M., and Paprzycki, M. Developing a model agent-based airline ticket auctioning system. In *Intelligent Information Processing and Web Mining*. Springer, 2006, pp. 297–306.

[173] Vytelingum, P., Ramchurn, S., Voice, T., Rogers, A., and Jennings, N. Agent-based modeling of smart-grid market operations. In *Power and Energy Society General Meeting, 2011 IEEE* (2011), IEEE, pp. 1–8.

[174] Wäldrich, O., Battre, D., Brazier, F., Clark, K., Oey, M., Papaspyrou, A., Wieder, P., and Ziegler, W. WS-Agreement Negotiation: Version 1.0 (GFD-R-P.193). Tech. rep., Open Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG, 2011.

[175] Wang, H., and Wang, Y. Multi-agent system negotiation based on expanded contract net protocol research. In *Natural Language Processing and Knowledge Engineering, 2005. IEEE NLP-KE'05. Proceedings of 2005 IEEE International Conference on* (2005), IEEE, pp. 472–478.

[176] Wang, X., and Schulzrinne, H. Rnap: A resource negotiation and pricing protocol. *Transit 6*, B7 (1999), B8.

[177] Wang, Y., and Lin, F.-r. Trust and risk evaluation of transactions with different amounts in peer-to-peer e-commerce environments. In *e-Business Engineering, 2006. ICEBE'06. IEEE International Conference on* (2006), IEEE, pp. 102–109.

[178] Wang, Y., Wong, D. S., Lin, K.-J., and Varadharajan, V. Evaluating transaction trust and risk levels in peer-to-peer e-commerce environments. *Information Systems and E-Business Management 6*, 1 (2008), 25–48.

[179] Wang, Y., and Zheng, Y. Fast and Secure Append-Only Storage with Infinite Capacity. In *Second IEEE International Security in Storage Workshop* (2003), IEEE, pp. 11–19.

[180] Weiss, A. Computing in the clouds. *netWorker 11*, 4 (2007), 16–25.

[181] Whitworth, B. Socio-technical systems. *Encyclopedia of human computer interaction* (2006), 533–541.

[182] Wijngaards, N., Van Steen, M., and Brazier, F. On mas scalability. In *Proc. 2nd Int'l Workshop on Infrastructure for Agents, MAS and Scalable MAS* (2001).

[183] Wijngaards, N. J., Overeinder, B., van Steen, M., and Brazier, F. M. Supporting internet-scale multi-agent systems. *Data & Knowledge Engineering 41*, 2 (2002), 229–245.

[184] Wong, T., and Fang, F. A multi-agent protocol for multilateral negotiations in supply chain management. *International Journal of Production Research 48*, 1 (2010), 271–299.

[185] Wooldridge, M. *An introduction to multiagent systems*. Wiley, 2008.

APPENDIX A

Supplemental Material of Chapter 5

```
/**
 * Context specific fields (e.g. minimum prices p/KWh)
 * src: Wikipedia, April 2013
 */
public enum EnergyPrice {

  NUCLEAR(0.1127),
  COAL(0.0996),
  GAS(1.1053),
  WIND(0.0968),
  SOLAR(0.1569),
  HYDRO(0.0899),
  BIOFUELS(0.1202),
  GEOTHERMAL(0.0996);
  private Double value;
  private EnergyPrice(Double d) {
    value = d;
  }
}
```

Figure A.1: Example energy provider minimum prices (in Java).

```
/**
 * Test if offered price of selected energy type is acceptable.
 */
public boolean evaluateNegotiationOffer(NegotiationOffer offer) {

 ServiceDescriptionTermType[] terms = offer.getTerms().getAll().
   getServiceDescriptionTermArray();
 for (ServiceDescriptionTermType sdt : terms) {

  // Go through each EnergyService term and check the price
  if (sdt.getServiceName().equals("EnergyService")) {
  EnergyDocument ed = EnergyDocument.Factory.parse(sdt.xmlText());
  EnergyType et = ed.getEnergy();
  if (et.getEnergySource().matches(EnergyPrice.WIND.toString())) {
   if (et.getEnergyPrice() < EnergyPrice.WIND.value) {

    // Suggested price for WIND is too low.
    return false;
   } else {

    // Suggested price for WIND is acceptable.
    return true;
   }
 ...
}
```

Figure A.2: Example energy provider offer evaluation (in Java).

```
/**
 *  Create counter-offer however you choose
 *  following a personal strategy, such as:
 *  if too low, increase by 10%
 *  if too high, decrease by 10%
 */
public NegotiationOffer createCounterOffer(NegotiationOffer offer,
  SessionInfo sessionInfo) {

 NegotiationOfferStateType state = NegotiationOfferStateType.Factory.
   newInstance();
 NegotiationOffer response = new NegotiationOffer(offer);

 // go through each term and, if unacceptable, propose something else
 ServiceDescriptionTermType[] terms = response.getTerms().getAll().
    getServiceDescriptionTermArray();
 for (ServiceDescriptionTermType sdt : terms) {
  if (sdt.getServiceName().equals("EnergyService")) {
   EnergyDocument ed = EnergyDocument.Factory.parse(sdt.xmlText());
   EnergyType et = ed.getEnergy();
    if (et.getEnergySource().matches(EnergyPrice.WIND.toString())) {
     if (et.getEnergyPrice() < EnergyPrice.WIND.value) {

      // Suggested WIND price too low. Increase 10% above minimum.
      double newPrice = EnergyPrice.WIND.value * 1.1;
      et.setEnergyPrice(newPrice);
     }
    ...

    // Put new values, if any, back into xml
    sdt.set(ed);
    sdt.setServiceName("EnergyService");
    ...

 // set the state to advisory or acceptable or solicited
 state.addNewAdvisory();
 response.getNegotiationOfferContext().setState(state);
 return response;
}
```

Figure A.3: Example energy provider counter-offer creation (in Java).

```
/**
 * Process an ACCEPTABLE offer. ACCEPT or create counter-offer.
 */
public NegotiationOffer processAcceptableOffer(NegotiationOffer offer,
  SessionInfo sessionInfo) {

 NegotiationOfferStateType state = NegotiationOfferStateType.Factory.
   newInstance();
 NegotiationOffer response = new NegotiationOffer(offer);

 // boolean test if offer is acceptable
 if (evaluateNegotiationOffer(offer)) {

  // Offer is acceptable.
  state.addNewAcceptable();
  response.getNegotiationOfferContext().setState(state);
 } else {

  // if offer is unacceptable, propose a counter-offer
  response = createCounterOffer(offer, sessionInfo);
 }
 return response;
}


/**
 * Process an ADVISORY offer. Essentially the same as processing an
 * ACCEPTABLE offer, but you can change it.
 */
public NegotiationOffer processAdvisoryOffer(NegotiationOffer offer,
  SessionInfo sessionInfo) {
 ...
}


/**
 * Process a REJECTED offer. If the counter-agent rejects your offer,
 * you can either end negotiation or rollback and create a new
 * counter-offer with better terms.
 */
public NegotiationOffer processRejectedOffer(NegotiationOffer offer,
  SessionInfo sessionInfo) {
 ...
}
```

Figure A.4: Helper methods for handling ACCEPTABLE, ADVISORY and REJECTED offers (in Java).

```
/**
 * Process a SOLICITED offer. You must respond with ACCEPT or REJECT.
 */
public NegotiationOffer processSolicitedOffer(NegotiationOffer offer,
  SessionInfo sessionInfo) {

 NegotiationOfferStateType state = NegotiationOfferStateType.Factory.
   newInstance();
 NegotiationOffer response = new NegotiationOffer(offer);

  // boolean test if offer is acceptable
  if (evaluateNegotiationOffer(offer)) {

   // Offer is acceptable. Set state to ACCEPTABLE.
   state.addNewAcceptable();
   response.getNegotiationOfferContext().setState(state);
  } else {

   // Offer is unacceptable! Set state to REJECTED.
   state.addNewRejected();
   response.getNegotiationOfferContext().setState(state);
  }
  return response;
}
```

Figure A.5: Helper method for handling SOLICITED offers (in Java).

```
/**
 * Context specific fields (e.g. maximum prices)
 * src: Wikipedia, April 2013
 * Willing to pay more for renewables, less for others
 */
public enum EnergyPrice {

  NUCLEAR(0.0564),
  COAL(0.0498),
  GAS(0.5527),
  WIND(0.1936),
  SOLAR(0.3138),
  HYDRO(0.1798),
  BIOFUELS(0.2404),
  GEOTHERMAL(0.1992);
  private Double value;
  private EnergyPrice(Double d) {
    value = d;
  }
}
```

Figure A.6: Example energy consumer maximum prices (in Java).

```
/**
 * Create a new offer, based on a template.
 * Follow a personal strategy. (e.g. low starting price)
 * Don't exceed creation constraints.
 */
public NegotiationOffer createNegotiationOffer(Template template,
  String sessionID) {

 NegotiationOffer offer = new NegotiationOffer(template, sessionID);

 // set session ID
 offer.setSessionID(sessionID);

 // prepare the negotiation context
 NegotiationOfferContextType context = NegotiationOfferContextType.
  Factory.newInstance();
 context.setCreator(NegotiationRoleType.NEGOTIATION_INITIATOR);
 ...

 // set the default state: acceptable, rejected, solicited, advisory
 NegotiationOfferStateType state = NegotiationOfferStateType.Factory.
  newInstance();
 state.addNewAdvisory();
 context.setState(state);

 // set counterOfferTo ID
 String counterOfferID = "template-" + template.getTemplateId();
 context.setCounterOfferTo(counterOfferID);

 // set the negotiation context
 offer.setNegotiationOfferContext(context);

 // use java to create the AgentScape specific XML terms
 EnergyDocument ed = EnergyDocument.Factory.newInstance();
 EnergyType et = ed.addNewEnergy();
 et.setEnergySource(EnergyPrice.WIND.toString());

 // choose an ideal price (e.g. very low)
 double idealPrice = EnergyPrice.WIND.value * 0.1;
 et.setEnergyPrice(idealPrice);

 // fill in all the terms
 ServiceDescriptionTermType[] terms = offer.getTerms().getAll().
   getServiceDescriptionTermArray();
 for (ServiceDescriptionTermType sdt : terms) {
  if (sdt.getServiceName().equals("EnergyService")) {
   sdt.set(ed);
   sdt.setName("RESOURCE_SDT");
   sdt.setServiceName("EnergyService");
  }
 }
return offer;
}
```

Figure A.7: Example energy consumer initial offer creation (in Java).

```
<NegotiationOffer OfferId="initiator-1">
  <Context>
    <AgreementProvider>AgreementResponder</AgreementProvider>
    <ServiceProvider>AgreementResponder</ServiceProvider>
    <TemplateId>4</TemplateId>
    <TemplateName>EnergyTemplate</TemplateName>
  </Context>
  <Terms>
    <All>
      <ServiceDescriptionTerm ServiceName="EnergyService">
        <Energy>
          <EnergySource>WIND</EnergySource>
          <EnergyPrice>0.019360000000000002</EnergyPrice>
        </Energy>
      </ServiceDescriptionTerm>
    </All>
  </Terms>
  <NegotiationOfferContext>
    <CounterOfferTo>template-4-EnergyTemplate</CounterOfferTo>
    <Creator>NegotiationInitiator</Creator>
    <State>
      <Advisory/>
    </State>
  </NegotiationOfferContext>
</NegotiationOffer>
```

Figure A.8: Energy consumer's initial negotiation offer: 'initiator-1'

```
<NegotiationOffer OfferId="responder-1">
  <Context>
    <AgreementProvider>AgreementResponder</AgreementProvider>
    <ServiceProvider>AgreementResponder</ServiceProvider>
    <TemplateId>4</TemplateId>
    <TemplateName>EnergyTemplate</TemplateName>
  </Context>
  <Terms>
    <All>
      <ServiceDescriptionTerm ServiceName="EnergyService">
        <Energy>
          <EnergySource>WIND</EnergySource>
          <EnergyPrice>0.10647999999999999</EnergyPrice>
        </Energy>
      </ServiceDescriptionTerm>
    </All>
  </Terms>
  <NegotiationOfferContext>
    <CounterOfferTo>initiator-1</CounterOfferTo>
    <Creator>NegotiationResponder</Creator>
    <State>
      <Advisory/>
    </State>
  </NegotiationOfferContext>
</NegotiationOffer>
```

Figure A.9: Energy provider's first counter-offer: 'responder-1'

```
<NegotiationOffer OfferId="initiator-2">
  <Context>
    <AgreementProvider>AgreementResponder</AgreementProvider>
    <ServiceProvider>AgreementResponder</ServiceProvider>
    <TemplateId>4</TemplateId>
    <TemplateName>EnergyTemplate</TemplateName>
  </Context>
  <Terms>
    <All>
      <ServiceDescriptionTerm ServiceName="EnergyService">
        <Energy>
          <EnergySource>WIND</EnergySource>
          <EnergyPrice>0.10647999999999999</EnergyPrice>
        </Energy>
      </ServiceDescriptionTerm>
    </All>
  </Terms>
  <NegotiationOfferContext>
    <CounterOfferTo>responder-1</CounterOfferTo>
    <Creator>NegotiationInitiator</Creator>
    <State>
      <Acceptable/>
    </State>
  </NegotiationOfferContext>
</NegotiationOffer>
```

Figure A.10: Energy consumer's acceptable offer: 'initiator-2'

```
<NegotiationOffer OfferId="responder-2">
  <Context>
    <AgreementProvider>AgreementResponder</AgreementProvider>
    <ServiceProvider>AgreementResponder</ServiceProvider>
    <TemplateId>4</TemplateId>
    <TemplateName>EnergyTemplate</TemplateName>
  </Context>
  <Terms>
    <All>
      <ServiceDescriptionTerm ServiceName="EnergyService">
        <Energy>
          <EnergySource>WIND</EnergySource>
          <EnergyPrice>0.10647999999999999</EnergyPrice>
        </Energy>
      </ServiceDescriptionTerm>
    </All>
  </Terms>
  <NegotiationOfferContext>
    <CounterOfferTo>initiator-2</CounterOfferTo>
    <Creator>NegotiationResponder</Creator>
    <State>
      <Acceptable/>
    </State>
  </NegotiationOfferContext>
</NegotiationOffer>
```

Figure A.11: Energy provider's acceptable offer: 'responder-2'

Index

List of Figures

List of Tables

Summary

Large scale, distributed, digital environments offer vast potential. Within these environments, software systems will provide unprecedented support for daily life. Offering access to vast amounts of knowledge and resources, these systems will enable wider participation of society, at large. An example is the Smart Energy Grid that increases sustainability and decreases reliance on fossil fuels.

Such systems require technology that is capable of negotiating Service Level Agreements (SLAs) between consumers and providers. Multi Agent Systems (MAS) is one such technology that offers a straightforward analog for complex systems of autonomous parties. MAS is based on the notion of autonomous agents that represent human actors (i.e. owners) and are capable of negotiating SLAs and coordinating processes with other agents. They know their owner's preferences and needs. They are capable of negotiating price, Quality of Service (QoS) characteristics and penalties. They also monitor provisioning of services to detect and penalize service violations.

This dissertation presents a MAS framework for automated negotiation and monitoring of SLAs in open environments. In this context, an open environment is a large-scale, distributed environment that is also dynamic and untrusted. This framework enables secure discovery, negotiation and access to distributed resources. Through a process of exchanging messages (e.g. offers, counter-offers), agents together search for a mutually acceptable agreement (e.g. service, price, quality). A negotiation protocol defines the negotiation objects (i.e. offers), language and rules governing interaction. This dissertation presents the WS-Agreement Negotiation protocol with extensions for open environments. This protocol is experimentally validated in the AgentScape middleware.

Open environments also present challenges regarding security, trust and privacy. No single authority has complete control over an open environment and no single authority governs the actions of all participants (i.e. agents). Therefore, additional mechanisms are required to ensure security, privacy and promote trust between participants. Automated monitoring mechanisms using

a Trusted Third Party (TTP) address issues of security and thus support negotiation in open environments. This dissertation presents a self-adaptive monitoring approach that (1) offers monitoring assurance that agreements are honored, (2) builds a secure audit log of agreement compliance, (3) performs measurements while safeguarding privacy of (sensitive) data, (4) dynamically reacts to changes in risk and (5) enables trust-building between consumers and providers. This monitoring approach is experimentally validated in the AgentScape middleware.

Automation of complex tasks, such as negotiation, can increase efficiency. To illustrate these benefits, the framework is applied to two complex systems, including the Smart Energy Grid. In this case, the looming complexity crisis of intermittent generation, real-time pricing and consumer demand shifting requires immediate attention. This domain presents not only technical (e.g. smart-meters) but also social challenges (e.g. user acceptance). The MAS automation framework presented in this dissertation addresses technical challenges by reducing manual labor and increasing efficiency. Automation even enables higher utilization of green resources and reduction of waste (e.g. produced, but unconsumed energy). Transparent, trusted monitoring mechanisms address social challenges by ensuring privacy of (sensitive) data and encouraging user acceptance. Software systems, such as those presented in this dissertation, enable wider participation of society, at large, and offer vast potential.

Samenvatting (Dutch summary)

Grootschalige, gedistribueerde, digitale omgevingen bieden veel mogelijkheden. Binnen dergelijke omgevingen zullen software systemen ongekende steun leveren aan het dagelijkse leven. Door toegang te bieden tot enorme hoeveelheden aan kennis en middelen, zullen deze systemen een bredere participatie van de samenleving mogelijk maken. Een voorbeeld van zo'n systeem is het slimme energie netwerk, oftewel Smart Energy Grid, dat duurzaamheid vergroot en afhankelijkheid van fossiele brandstoffen verminderd.

Dergelijke systemen vereisen technologie die in staat is serviceniveau overeenkomsten, oftewel Service Level Agreements (SLA), tussen consumenten en leveranciers te onderhandelen. Multi Agent Systems (MAS) is een geschikte technologie die een eenvoudige analogie biedt voor complexe systemen met autonome deelnemers. MAS is gebaseerd op het concept van autonome agenten die menselijke actoren (d.w.z. eigenaren) vertegenwoordigen en in staat zijn om SLA's te onderhandelen en processen met andere agenten te coördineren. Zij kennen de voorkeuren en behoeften van hun eigenaar. Zij zijn in staat om te onderhandelen over prijs, kwaliteit van diensten, attributen en sancties. Zij zijn ook in staat om dienstlevering te monitoren om overtredingen te detecteren en bestraffen.

Dit proefschrift presenteert een MAS raamwerk voor geautomatiseerde onderhandelen en monitoren van SLA's in open omgevingen. In deze context is een open omgeving een grootschalig, gedistribueerde omgeving, die zowel dynamisch als onbetrouwbaar is. Dit raamwerk biedt beveiligde detectie, onderhandeling en toegang tot gedistribueerde middelen. Door middel van een proces waarin berichten worden uitgewisseld (d.w.z. bod, tegenbod) zoeken agenten voor een wederzijds aanvaardbare overeenkomst (d.w.z. diensten, prijs, kwaliteit). Een onderhandelingsprotocol definieert de onderhandelingsvoorwerpen (d.w.z. bod), taal en regels voor interactie. Dit proefschrift presenteert het WS-Agreement Negotiation protocol met uitbreidingen voor open omgevingen. Dit protocol wordt experimenteel gevalideerd in de AgentScape middleware.

Open omgevingen zorgen ook voor uitdagingen rondom veiligheid, vertrouwen en privacy. Geen enkele autoriteit heeft volledige controle over een open omgeving en geen enkele autoriteit bepaalt welke acties deelnemers (d.w.z. agenten) mogen uitvoeren. Daarom zijn aanvullende mechanismen nodig om veiligheid en privacy te garanderen, en het vertrouwen tussen deelnemers te bevorderen. Geautomatiseerde monitoring mechanismen, die gebruik maken van een vertrouwde derde partij, richten zich op veiligheid en maken daardoor onderhandeling in open omgevingen mogelijk. Dit proefschrift presenteert een zelf-adaptieve monitoring aanpak welke (1) zekerheid biedt dat afspraken worden nageleefd, (2) een veilige audit log bouwt, (3) metingen uitvoert zonder privacy van (gevoelige) data te schaden, (4) dynamisch reageert op veranderende risico's en (5) het opbouwen van het vertrouwen tussen consumenten en leveranciers faciliteert. Deze monitoring aanpak wordt experimenteel gevalideerd in de AgentScape middleware.

Het automatiseren van complexe taken, zoals onderhandelingen, kan efficiëntie vergroten. Om deze voordelen te illustreren, wordt dit raamwerk toegepast op twee complexe systemen, waaronder het slimme energie netwerk. In dit geval, vereist de dreigende complexiteitscrisis van intermitterende energiebronnen, actuele prijzen en consumenten vraagverschuiving onmiddellijke aandacht. Dit domein presenteert niet alleen technisch (bijv. slimme elektriciteitsmeter) maar ook sociale uitdagingen (bijv. gebruikersacceptatie). Het MAS raamwerk dat in dit proefschrift gepresenteerd wordt, richt zich op de technische uitdagingen door het verminderen van handenarbeid en het vergroten van efficiëntie. Het automatiseren maakt zelfs hogere benutting van groene energiebronnen en vermindering van verspilling mogelijk. Transparant, vertrouwde monitoringmechanismen richten zich op de sociale uitdagingen door privacy van (gevoelige) data te waarborgen en gebruikersacceptatie te bemoedigen. Software systemen, zoals deze in dit proefschrift worden gepresenteerd, maken bredere participatie van de samenleving mogelijk en bieden veel mogelijkheden.

SIKS Dissertation Series

2009-01 Rasa Jurgelenaite (RUN)

Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU)

Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT)

A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN)

Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietse Overbeek (RUN)

Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU)

Understanding Classification

2009-07 Ronald Poppe (UT)

Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU)

Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN)

Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UVA)

Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UVA)

Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)

Operating Guidelines for Services

2009-13 Steven de Jong (UM)

Fairness in Multi-Agent Systems

2009-14 Maksym Korotkiy (VU)

From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA)

Ontology Representation - Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT)

New Architectures in Computer Chess

2009-17 Laurens van der Maaten (UvT)

Feature Extraction from Visual Data

2009-18 Fabian Groffen (CWI)

Armada, An Evolving Database System

2009-19 Valentin Robu (CWI)

Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU)

Adjustable Autonomy: Controling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM)

Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT)

Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU)

Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA)

Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI)

RAM: Array Database Management through Relational Mapping

2009-26 Fernando Koch (UU)

An Agent-Based Model for the Development of Intelligent Mobile Services

2009-27 Christian Glahn (OU)

Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT)

Sensor Data Management with Probabilistic Models

2009-29 Stanislav Pokraev (UT)

Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI)

Balancing vectorized query execution with bandwidth-optimized storage

2009-31 Sofiya Katrenko (UVA)

A Closer Look at Learning Relations from Text

2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)

Architectural Knowledge Management: Supporting Architects and Auditors

2009-33 Khiet Truong (UT)

How Does Real Affect Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU)

Advancing in Software Product Management: An Incremental Method Engineering Approach

2009-35 Wouter Koelewijn (UL)

Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling

2009-36 Marco Kalz (OUN)

Placement Support for Learners in Learning Networks

2009-37 Hendrik Drachsler (OUN)

Navigation Support for Learners in Informal Learning Networks

2009-38 Riina Vuorikari (OU)

Tags and self-organisation: a metadata ecology for learning resources in a multilingual context

2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)

Service Substitution – A Behavioral Approach Based on Petri Nets

2009-40 Stephan Raaijmakers (UvT)

Multinomial Language Learning: Investigations into the Geometry of Language

2009-41 Igor Berezhnyy (UvT)

Digital Analysis of Paintings

2009-42 Toine Bogers (UvT)

Recommender Systems for Social Bookmarking

2009-43 Virginia Nunes Leal Franqueira (UT)

Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients

2009-44 Roberto Santana Tapia (UT)

Assessing Business-IT Alignment in Networked Organizations

2009-45 Jilles Vreeken (UU)

Making Pattern Mining Useful

2009-46 Loredana Afanasiev (UvA)

Querying XML: Benchmarks and Recursion

2010

2010-01 Matthijs van Leeuwen (UU)

Patterns that Matter

2010-02 Ingo Wassink (UT)

Work flows in Life Science

2010-03 Joost Geurts (CWI)

A Document Engineering Model and Processing Framework for Multimedia documents

2010-04 Olga Kulyk (UT)

Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments

2010-05 Claudia Hauff (UT)

Predicting the Effectiveness of Queries and Retrieval Systems

2010-06 Sander Bakkes (UvT)

Rapid Adaptation of Video Game AI

2010-07 Wim Fikkert (UT)

A Gesture interaction at a Distance

2010-08 Krzysztof Siewicz (UL)

Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments

2010-09 Hugo Kielman (UL)

A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging

2010-10 Rebecca Ong (UL)

Mobile Communication and Protection of Children

2010-11 Adriaan Ter Mors (TUD)

The world according to MARP: Multi-Agent Route Planning

2010-12 Susan van den Braak (UU)

Sensemaking software for crime analysis

2010-13 Gianluigi Folino (RUN)

High Performance Data Mining using Bio-inspired techniques

2010-14 Sander van Splunter (VU)

Automated Web Service Reconfiguration

2010-15 Lianne Bodenstaff (UT)

Managing Dependency Relations in Inter-Organizational Models

2010-16 Sicco Verwer (TUD)

Efficient Identification of Timed Automata, theory and practice

2010-17 Spyros Kotoulas (VU)

Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications

2010-18 Charlotte Gerritsen (VU)

Caught in the Act: Investigating Crime by Agent-Based Simulation

2010-19 Henriette Cramer (UvA)

People's Responses to Autonomous and Adaptive Systems

2010-20 Ivo Swartjes (UT)

Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative

2010-21 Harold van Heerde (UT)

Privacy-aware data management by means of data degradation

2010-22 Michiel Hildebrand (CWI)

End-user Support for Access to Heterogeneous Linked Data

2010-23 Bas Steunebrink (UU)

The Logical Structure of Emotions

2010-24 Dmytro Tykhonov

Designing Generic and Efficient Negotiation Strategies

2010-25 Zulfiqar Ali Memon (VU)

Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective

2010-26 Ying Zhang (CWI)

XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines

2010-27 Marten Voulon (UL)

Automatisch contracteren

2010-28 Arne Koopman (UU)

Characteristic Relational Patterns

2010-29 Stratos Idreos (CWI)

Database Cracking: Towards Auto-tuning Database Kernels

2010-30 Marieke van Erp (UvT)

Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval

2010-31 Victor de Boer (UVA)

Ontology Enrichment from Heterogeneous Sources on the Web

2010-32 Marcel Hiel (UvT)

An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems

2010-33 Robin Aly (UT)

Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval

2010-34 Teduh Dirgahayu (UT)

Interaction Design in Service Compositions

2010-35 Dolf Trieschnigg (UT)

Proof of Concept: Concept-based Biomedical Information Retrieval

2010-36 Jose Janssen (OU)

Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification

2010-37 Niels Lohmann (TUE)

Correctness of services and their composition

2010-38 Dirk Fahland (TUE)

From Scenarios to components

2010-39 Ghazanfar Farooq Siddiqui (VU)

Integrative modeling of emotions in virtual agents

2010-40 Mark van Assem (VU)

Converting and Integrating Vocabularies for the Semantic Web

2010-41 Guillaume Chaslot (UM)

Monte-Carlo Tree Search

2010-42 Sybren de Kinderen (VU)

Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach

2010-43 Peter van Kranenburg (UU)

A Computational Approach to Content-Based Retrieval of Folk Song Melodies

2010-44 Pieter Bellekens (TUE)

An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain

2010-45 Vasilios Andrikopoulos (UvT)

A theory and model for the evolution of software services

2010-46 Vincent Pijpers (VU)

e3alignment: Exploring Inter-Organizational Business-ICT Alignment

2010-47 Chen Li (UT)

Mining Process Model Variants: Challenges, Techniques, Examples

2010-48 Withdrawn

2010-49 Jahn-Takeshi Saito (UM)

Solving difficult game positions

2010-50 Bouke Huurnink (UVA)

Search in Audiovisual Broadcast Archives

2010-51 Alia Khairia Amin (CWI)

Understanding and supporting information seeking tasks in multiple sources

2010-52 Peter-Paul van Maanen (VU)

Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention

2010-53 Edgar Meij (UVA)

Combining Concepts and Language Models for Information Access

2011

2011-01 Botond Cseke (RUN)

Variational Algorithms for Bayesian Inference in Latent Gaussian Models

2011-02 Nick Tinnemeier(UU)

Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

2011-03 Jan Martijn van der Werf (TUE)

Compositional Design and Verification of Component-Based Information Systems

2011-04 Hado van Hasselt (UU)

Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms

2011-05 Base van der Raadt (VU)

Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline

2011-06 Yiwen Wang (TUE)

Semantically-Enhanced Recommendations in Cultural Heritage

2011-07 Yujia Cao (UT)

Multimodal Information Presentation for High Load Human Computer Interaction

2011-08 Nieske Vergunst (UU)

BDI-based Generation of Robust Task-Oriented Dialogues

2011-09 Tim de Jong (OU)

Contextualised Mobile Media for Learning

2011-10 Bart Bogaert (UvT)

Cloud Content Contention

2011-11 Dhaval Vyas (UT)

Designing for Awareness: An Experience-focused HCI Perspective

2011-12 Carmen Bratosin (TUE)

Grid Architecture for Distributed Process Mining

2011-13 Xiaoyu Mao (UvT)

Airport under Control. Multiagent Scheduling for Airport Ground Handling

2011-14 Milan Lovric (EUR)

Behavioral Finance and Agent-Based Artificial Markets

2011-15 Marijn Koolen (UvA)

The Meaning of Structure: the Value of Link Evidence for Information Retrieval

2011-16 Maarten Schadd (UM)

Selective Search in Games of Different Complexity

2011-17 Jiyin He (UVA)

Exploring Topic Structure: Coherence, Diversity and Relatedness

2011-18 Mark Ponsen (UM)

Strategic Decision-Making in complex games

2011-19 Ellen Rusman (OU)

The Mind's Eye on Personal Profiles

2011-20 Qing Gu (VU)

Guiding service-oriented software engineering - A view-based approach

2011-21 Linda Terlouw (TUD)

Modularization and Specification of Service-Oriented Systems

2011-22 Junte Zhang (UVA)

System Evaluation of Archival Description and Access

2011-23 Wouter Weerkamp (UVA)

Finding People and their Utterances in Social Media

2011-24 Herwin van Welbergen (UT)

Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

2011-25 Syed Waqar ul Qounain Jaffry (VU)

Analysis and Validation of Models for Trust Dynamics

2011-26 Matthijs Aart Pontier (VU)

Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots

2011-27 Aniel Bhulai (VU)

Dynamic website optimization through autonomous management of design patterns

2011-28 Rianne Kaptein(UVA)

Effective Focused Retrieval by Exploiting Query Context and Document Structure

2011-29 Faisal Kamiran (TUE)

Discrimination-aware Classification

2011-30 Egon van den Broek (UT)

Affective Signal Processing (ASP): Unraveling the mystery of emotions

2011-31 Ludo Waltman (EUR)

Computational and Game-Theoretic Approaches for Modeling Bounded Rationality

2011-32 Nees-Jan van Eck (EUR)

Methodological Advances in Bibliometric Mapping of Science

2011-33 Tom van der Weide (UU)

Arguing to Motivate Decisions

2011-34 Paolo Turrini (UU)

Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

2011-35 Maaike Harbers (UU)

Explaining Agent Behavior in Virtual Training

2011-36 Erik van der Spek (UU)

Experiments in serious game design: a cognitive approach

2011-37 Adriana Burlutiu (RUN)

Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference

2011-38 Nyree Lemmens (UM)

Bee-inspired Distributed Optimization

2011-39 Joost Westra (UU)

Organizing Adaptation using Agents in Serious Games

2011-40 Viktor Clerc (VU)

Architectural Knowledge Management in Global Software Development

2011-41 Luan Ibraimi (UT)

Cryptographically Enforced Distributed Data Access Control

2011-42 Michal Sindlar (UU)

Explaining Behavior through Mental State Attribution

2011-43 Henk van der Schuur (UU)

Process Improvement through Software Operation Knowledge

2011-44 Boris Reuderink (UT)

Robust Brain-Computer Interfaces

2011-45 Herman Stehouwer (UvT)

Statistical Language Models for Alternative Sequence Selection

2011-46 Beibei Hu (TUD)

Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work

2011-47 Azizi Bin Ab Aziz (VU)

Exploring Computational Models for Intelligent Support of Persons with Depression

2011-48 Mark Ter Maat (UT)

Response Selection and Turn-taking for a Sensitive Artificial Listening Agent

2011-49 Andreea Niculescu (UT)

Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

2012

2012-01 Terry Kakeeto (UvT)

Relationship Marketing for SMEs in Uganda

2012-02 Muhammad Umair (VU)

Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

2012-03 Adam Vanya (VU)

Supporting Architecture Evolution by Mining Software Repositories

2012-04 Jurriaan Souer (UU)

Development of Content Management System-based Web Applications

2012-05 Marijn Plomp (UU)

Maturing Interorganisational Information Systems

2012-06 Wolfgang Reinhardt (OU)

Awareness Support for Knowledge Workers in Research Networks

2012-07 Rianne van Lambalgen (VU)

When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions

2012-08 Gerben de Vries (UVA)

Kernel Methods for Vessel Trajectories

2012-09 Ricardo Neisse (UT)

Trust and Privacy Management Support for Context-Aware Service Platforms

2012-10 David Smits (TUE)

Towards a Generic Distributed Adaptive Hypermedia Environment

2012-11 J.C.B. Rantham Prabhakara (TUE)

Process Mining in the Large: Preprocessing, Discovery, and Diagnostics

2012-12 Kees van der Sluijs (TUE)

Model Driven Design and Data Integration in Semantic Web Information Systems

2012-13 Suleman Shahid (UvT)

Fun and Face: Exploring non-verbal expressions of emotion during playful interactions

2012-14 Evgeny Knutov (TUE)

Generic Adaptation Framework for Unifying Adaptive Web-based Systems

2012-15 Natalie van der Wal (VU)

Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes

2012-16 Fiemke Both (VU)

Helping people by understanding them - Ambient Agents supporting task execution and depression treatment

2012-17 Amal Elgammal (UvT)

Towards a Comprehensive Framework for Business Process Compliance

2012-18 Eltjo Poort (VU)

Improving Solution Architecting Practices

2012-19 Helen Schonenberg (TUE)

What's Next? Operational Support for Business Process Execution

2012-20 Ali Bahramisharif (RUN)

Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing

2012-21 Roberto Cornacchia (TUD)

Querying Sparse Matrices for Information Retrieval

2012-22 Thijs Vis (UvT)

Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

2012-23 Christian Muehl (UT)

Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

2012-24 Laurens van der Werff (UT)

Evaluation of Noisy Transcripts for Spoken Document Retrieval

2012-25 Silja Eckartz (UT)

Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application

2012-26 Emile de Maat (UVA)

Making Sense of Legal Text

2012-27 Hayrettin Gurkok (UT)

Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games

2012-28 Nancy Pascall (UvT)

Engendering Technology Empowering Women

2012-29 Almer Tigelaar (UT)

Peer-to-Peer Information Retrieval

2012-30 Alina Pommeranz (TUD)

Designing Human-Centered Systems for Reflective Decision Making

2012-31 Emily Bagarukayo (RUN)

A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure

2012-32 Wietske Visser (TUD)

Qualitative multi-criteria preference representation and reasoning

2012-33 Rory Sie (OUN)

Coalitions in Cooperation Networks (COCOON)

2012-34 Pavol Jancura (RUN)

Evolutionary analysis in PPI networks and applications

2012-35 Evert Haasdijk (VU)

Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics

2012-36 Denis Ssebugwawo (RUN)

Analysis and Evaluation of Collaborative Modeling Processes

2012-37 Agnes Nakakawa (RUN)

A Collaboration Process for Enterprise Architecture Creation

2012-38 Selmar Smit (VU)

Parameter Tuning and Scientific Testing in Evolutionary Algorithms

2012-39 Hassan Fatemi (UT)

Risk-aware design of value and coordination networks

2012-40 Agus Gunawan (UvT)

Information Access for SMEs in Indonesia

2012-41 Sebastian Kelle (OU)

Game Design Patterns for Learning

2012-42 Dominique Verpoorten (OU)

Reflection Amplifiers in self-regulated Learning

2012-43 Withdrawn

2012-44 Anna Tordai (VU)

On Combining Alignment Techniques

2012-45 Benedikt Kratz (UvT)

A Model and Language for Business-aware Transactions

2012-46 Simon Carter (UVA)

Exploration and Exploitation of Multilingual Data for Statistical Machine Translation

2012-47 Manos Tsagkias (UVA)

Mining Social Media: Tracking Content and Predicting Behavior

2012-48 Jorn Bakker (TUE)

Handling Abrupt Changes in Evolving Time-series Data

2012-49 Michael Kaisers (UM)

Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions

2012-50 Steven van Kervel (TUD)

Ontologogy driven Enterprise Information Systems Engineering

2012-51 Jeroen de Jong (TUD)

Heuristics in Dynamic Scheduling; a practical framework with a case study in elevator dispatching

2013

2013-01 Viorel Milea (EUR)

News Analytics for Financial Decision Support

2013-02 Erietta Liarou (CWI)

MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing

2013-03 Szymon Klarman (VU)

Reasoning with Contexts in Description Logics

2013-04 Chetan Yadati (TUD)

Coordinating autonomous planning and scheduling

2013-05 Dulce Pumareja (UT)

Groupware Requirements Evolutions Patterns

2013-06 Romulo Gonzalves (CWI)

The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience

2013-07 Giel van Lankveld (UT)

Quantifying Individual Player Differences

2013-08 Robbert-Jan Merk (VU)

Making enemies: cognitive modeling for opponent agents in fighter pilot simulators

2013-09 Fabio Gori (RUN)

Metagenomic Data Analysis: Computational Methods and Applications

2013-10 Jeewanie Jayasinghe Arachchige (UvT)

A Unified Modeling Framework for Service Design

2013-11 Evangelos Pournaras (TUD)

Multi-level Reconfigurable Self-organization in Overlay Services

2013-12 Marian Razavian (VU)

Knowledge-driven Migration to Services

2013-13 Mohammad Safiri (UT)

Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly

2013-14 Jafar Tanha (UVA)

Ensemble Approaches to Semi-Supervised Learning Learning

2013-15 Daniel Hennes (UM)

Multiagent Learning - Dynamic Games and Applications

2013-16 Eric Kok (UU)

Exploring the practical benefits of argumentation in multi-agent deliberation

2013-17 Koen Kok (VU)

The PowerMatcher: Smart Coordination for the Smart Electricity Grid

2013-18 Jeroen Janssens (UvT)

Outlier Selection and One-Class Classification

2013-19 Renze Steenhuizen (TUD)

Coordinated Multi-Agent Planning and Scheduling

2013-20 Katja Hofmann (UvA)

Fast and Reliable Online Learning to Rank for Information Retrieval

2013-21 Sander Wubben (UvT)

Text-to-text generation by monolingual machine translation

2013-22 Tom Claassen (RUN)

Causal Discovery and Logic

2013-23 Patricio de Alencar Silva (UvT)

Value Activity Monitoring

2013-24 Haitham Bou Ammar (UM)

Automated Transfer in Reinforcement Learning

2013-25 Agnieszka Anna Latoszek-Berendsen (UM)

Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System

2013-26 Alireza Zarghami (UT)

Architectural Support for Dynamic Homecare Service Provisioning

2013-27 Mohammad Huq (UT)

Inference-based Framework Managing Data Provenance

2013-28 Frans van der Sluis (UT)

When Complexity becomes Interesting: An Inquiry into the Information eXperience

2013-29 Iwan de Kok (UT)

Listening Heads

2013-30 Joyce Nakatumba (TUE)

Resource-Aware Business Process Management: Analysis and Support

2013-31 Dinh Khoa Nguyen (UvT)

Blueprint Model and Language for Engineering Cloud Applications

2013-32 Kamakshi Rajagopal (OUN)

Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development

2013-33 Qi Gao (TUD)

User Modeling and Personalization in the Microblogging Sphere

2013-34 Kien Tjin-Kam-Jet (UT)

Distributed Deep Web Search

2013-35 Abdallah El Ali (UvA)

Minimal Mobile Human Computer Interaction

2013-36 Than Lam Hoang (TUe)

Pattern Mining in Data Streams

2013-37 Dirk BÃűrner (OUN)

Ambient Learning Displays

2013-38 Eelco den Heijer (VU)

Autonomous Evolutionary Art

2013-39 Joop de Jong (TUD)

A Method for Enterprise Ontology based Design of Enterprise Information Systems

2013-40 Pim Nijssen (UM)

Monte-Carlo Tree Search for Multi-Player Games

2013-41 Jochem Liem (UVA)

Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning

2013-42 LÃľon Planken (TUD)

Algorithms for Simple Temporal Reasoning

2013-43 Marc Bron (UVA)

Exploration and Contextualization through Interaction and Concepts

2014

2014-01 Nicola Barile (UU)

Studies in Learning Monotone Models from Data

2014-02 Fiona Tuliyano (RUN)

Combining System Dynamics with a Domain Modeling Method

2014-03 Sergio Raul Duarte Torres (UT)

Information Retrieval for Children: Search Behavior and Solutions

2014-04 Hanna Jochmann-Mannak (UT)

Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation

2014-05 Jurriaan van Reijsen (UU)

Knowledge Perspectives on Advancing Dynamic Capability

2014-06 Damian Tamburri (VU)

Supporting Networked Software Development

2014-07 Arya Adriansyah (TUE)

Aligning Observed and Modeled Behavior

2014-08 Samur Araujo (TUD)

Data Integration over Distributed and Heterogeneous Data Endpoints

2014-09 Philip Jackson (UvT)

Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language

2014-10 Ivan Salvador Razo Zapata (VU)

Service Value Networks

2014-11 Janneke van der Zwaan (TUD)

An Empathic Virtual Buddy for Social Support

2014-12 Willem van Willigen (VU)

Look Ma, No Hands: Aspects of Autonomous Vehicle Control

2014-13 Arlette van Wissen (VU)

Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains

2014-14 Yangyang Shi (TUD)

Language Models With Meta-information

2014-15 Natalya Mogles (VU)

Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare

2014-16 Krystyna Milian (VU)

Supporting trial recruitment and design by automatically interpreting eligibility criteria

2014-17 Kathrin Dentler (VU)

Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability

2014-18 Mattijs Ghijsen (VU)

Methods and Models for the Design and Study of Dynamic Agent Organizations

2014-19 Vincius Ramos (TUE)

Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support

2014-20 Mena Habib (UT)

Named Entity Extraction and Disambiguation for Informal Text: The Missing Link

2014-21 Kassidy Clark (TUD)

Negotiation and Monitoring in Open Environments

Publications

Clark, K. P., Warnier, M., and Brazier, F. M. T. Automated non-repudiable cloud resource allocation. In *Cloud Computing and Services Science*. Springer, 2013, pp. 168–182.

Clark, K. P., Warnier, M., and Brazier, F. M. T. Self-Adaptive Service Level Agreement Monitoring in Cloud Environments. *Multiagent and Grid Systems 9* (2013).

Clark, K. P., Warnier, M., and Brazier, F. M. T. Increasing green energy market efficiency using micro agreements. In *Green ICT & Energy: From Smart to Wise Strategies*, J. H. Appelman, A. Osseyran, and M. Warnier, Eds., Sustainable Energy Developments. CRC Press, 2013, pp. 77–91.

Clark, K. P., Warnier, M., and Brazier, F. M. T. An intelligent cloud resource allocation service - agent-based automated cloud resource allocation using micro-agreements. In *In the proceedings of the 2nd International Conference on Cloud Computing and Services Science (CLOSER 2012)* (2012), pp. 37–45.

Clark, K. P., Warnier, M., and Brazier, F. M. T. Self-adaptive service monitoring. In *proceedings of the 2011 International Conference on Adaptive and Intelligent Systems (ICAIS 2011)* (2011), Springer, pp. 119–130.

Wäldrich, O., Battre, D., Brazier, F. M. T., Clark, K. P., Oey, M. A., Papaspyrou, A., Wieder, P., and Ziegler, W. WS-Agreement Negotiation: Version 1.0 (GFD-R-P.193). Tech. rep., Open Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG, 2011.

Clark, K. P., Warnier, M., Brazier, F. M. T. BOTCLOUDS - The Future of Cloud-based Botnets?. In *proceedings of the 1st International*

Conference on Cloud Computing and Services Sciences (CLOSER 2011) (2011), pp. 597–603.

Clark, K. P., Warnier, M., Quillinan, T. B., and Brazier, F. M. T. Secure monitoring of service level agreements. In *Fifth International Conference on Availability, Reliability and Security (ARES 2010)* (March 2010), IEEE, pp. 454–461.

Clark, K. P., van Splunter, S., Warnier, M., and Brazier, F. M. T. Expressing intervals in automated service negotiation. In *Grids and Service-Oriented Architectures for Service Level Agreements*, P. Wieder, R. Yahyapour, and W. Ziegler, Eds., CoreGRID. Springer-Verlag, New York, NY, USA, 2010, pp. 67–76.

Battre, D., Brazier, F. M. T., Clark, K. P., Oey, M. A., Papaspyrou, A., Wäldrich, O., Wieder, P., and Ziegler, W. A proposal for WS-agreement negotiation. In *11th IEEE/ACM International Conference on Grid Computing* (2010).

Quillinan, T. B., Clark, K. P., Warnier, M., Brazier, F. M. T., and Rana, O. Negotiation and monitoring of service level agreements. In *Grids and Service-Oriented Architectures for Service Level Agreements*, P. Wieder, R. Yahyapour, and W. Ziegler, Eds., CoreGRID. Springer-Verlag, New York, NY, USA, 2010, pp. 167–176.

Clark, K. P. Automated security classification. Master's thesis, Vrije Universiteit Amsterdam, 2008.

Curriculum Vitae

Kassidy Patrick Clark was born in Wiesbaden, Germany on 24 June 1980. He spent most of his youth in Europe before moving to the United States for high school and university. He completed his Bachelor's degree *cum laude* at St. Edward's University in Austin, Texas with a major in Computer Science. His thesis focused on measuring the 'diameter' of the Internet. In addition, he studied a dual minor of German and theology in collaboration with the University of Koblenz in Germany, where he spent the final year of his undergraduate work.

In 2003, he moved to the Netherlands and continued his studies at the Vrije Universiteit Amsterdam where he earned a Master's degree in Computer Science with an emphasis on networking and security. His thesis focused on automated security classification, which applied techniques from Artificial Intelligence to classify documents according to the security rating (i.e. sensitivity) of their content. Through this research, he came into contact with the Intelligent Interactive Distributed Systems (IIDS) group, led by Professor Frances Brazier, where he started his Ph.D. research. In 2009, the IIDS group moved to the Delft University of Technology to join the Systems Engineering group at the Faculty of Technology, Policy and Management.

His Ph.D. research focused on designing and implementing protocols for automated negotiation of services in large scale, dynamic, distributed environments. Furthermore, his work addressed issues of trust and security in these environments through adaptive monitoring policies. This research has been presented at international conferences in the Netherlands, Canada, Poland, Germany, Austria, Portugal and Ireland. In addition, it has produced 11 peer-reviewed publications including an official Open Grid Forum standard for automated negotiation of web services. This work has also been published in several industry periodicals.