Some Service Issues in Adaptable Service Systems

Shanshan Jiang

Doctoral Thesis

Submitted for the Partial Fulfilment of the Requirements for the Degree of

Doktor Ingeniør

Department of Telematics
Faculty of Information Technology, Mathematics and Electrical Engineering
Norwegian University of Science and Technology

February 2008

NTNU
Norwegian University of Science and Technology

Thesis for the degree Doktor Ingeniør

Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Telematics

ISBN 978-82-471-6955-1 (printed version)
ISBN 978-82-471-6969-8 (electronic version)
ISSN 1503-8181
Doctoral theses at NTNU, 2008:52

Printed in Norway by NTNU Trykk, Trondheim

Abstract

Networked services have been an important research topic for over 40 years. These days, the amount and variety of services are growing enormously at the same time as the complexity and heterogeneity of the service systems is also increasing. Adaptable services and service systems are a research issue aiming to cope with the complexity.

Adaptable Service Systems are service systems that are able to adapt dynamically to changes in time and position related to users, nodes, capabilities, status, changed service requirements and policies.

A *service* can be considered at different abstraction levels. In this thesis, three abstraction levels are used, denoted as the conceptual, engineered and physical services. *Service engineering* is the creation of conceptual, engineered and physical services. *Service management* is the functionality to control the provision of service functionality and quality of a service, both within and across service systems, through the service life cycle phases.

This thesis addresses some service issues related to service engineering and service management in adaptable service systems. The work presented in this thesis is related with TAPAS (Telematics Architecture for Play-based Adaptable Service Systems). On one hand, TAPAS concepts, architectures and platform are the context and the basis of the thesis. On the other hand, my research work also aims to further develop TAPAS concepts, architectures and platform. The research aims to answer the following five problem statements:

P1: How can services be modelled and represented?
P2: How can services be discovered efficiently, automatically and accurately?
P3: How can services be instantiated dynamically and according to available capabilities and status information?
P4: How can new service specifications or modifications to existing services dynamically be introduced without interrupting the executing services?
P5: How to evaluate and validate the proposed frameworks and mechanisms?

The problem statements P1-P4 are related to the following four research topics:

T1: Service representation
T2: Service discovery
T3: Service instantiation
T4: Service adaptation.

Service representation is the representation of a service (conceptual, engineered and physical) based on a specific language and a data model. *Service discovery* is the process of finding services that satisfy functional and non-functional requirements. It is a core functionality to locate desired services in a distributed environment. *Service instantiation* is the process of creating a service instance upon request and making it available to the user, and finally *service adaptation* is the process of adapting the structure or behaviour of the service to the various changes during its execution.

There is one-to-one mapping from P1-P4 to T1-T4. P5 is related with all the four research topics T1-T4. The problem statements P1 and P2 are further refined into sub-problems. The problem statement P1 is refined into sub-problem statements P1.1-P1.3 defined as follows:

P1.1: How to represent conceptual services?
P1.2: How to represent physical services in a flexible manner so that it is possible to adapt the services to changes dynamically?
P1.3: How to extract the component interface behaviour from the physical service representation so that compositional service verification can be applied?

The problem statement P2 is refined into sub-problem statements P2.1-P2.2 defined as follows:

P2.1: How to ensure automatic and accurate service discovery?
P2.2: How to locate services efficiently in a large-scale service system?

The result of the research work is classified as nine *research contributions* C1-C9. These contributions are related to the *research topics* and accordingly *problem statements* as defined below:

Research topic T1 Service representation:

- ***C1:*** *Conceptual service representation.* This contribution addresses **P1.1**. An integrated semantic service description based on a service ontology is proposed and is represented using Web Services and Semantic Web languages. The service ontology defines a model of functional and non-functional properties, where the service functionality is represented as operations, inputs, outputs, preconditions and effects and the non-functional properties include service parameters, Quality of Service (QoS) parameters and policies consisting of business policies, QoS policies and context policies. Such semantic-annotated service description is the basis for semantic matching procedure in service discovery.
- ***C2:*** *Physical service representation.* This contribution addresses **P1.2**. XML (eXtensible Markup Language) is the physical service representation language. An Extended Finite State Machine (EFSM)-based XML manuscript data model is defined. It is based on modifiable and parameterized behaviour patterns, separating action types from actual action codes. Service functionality is further classified into Action Groups and Capability Categories according to the nature of actions and the dependability on capability respectively. Such manuscript data model is the basis for service instantiation and adaptation.
- ***C3:*** *Preparation for service verification*. This contribution addresses **P1.3**. *Service verification* is the process of checking service specifications to ensure that service components can play well together. In order to utilize compositional verification based on an interface type language, rules are given for automatic translation from EFSM-based XML manuscript to the interface type language. Projection technique is applied during the translation process.

Research topic T2 Service discovery:

- ***C4:*** *Semantic service discovery procedure.* This contribution addresses **P2.1**. An integrated semantic service description model is defined based on a service ontology (i.e. the conceptual service representation). An integrated semantic discovery procedure based on such service descriptions is proposed for semantic matching of both functional and non-functional properties. Such procedure consists of both ontological inference and rule-based reasoning and has been implemented on a Reasoning Machine (RM).
- ***C5:*** *Super-peer Semantic Overlay Network (SON)-based service discovery system.* This contribution addresses **P2.2**. A service discovery system based on super-peer managed SONs is proposed and functionality for efficient service discovery and efficient SON management is defined. The integrated semantic service discovery procedure proposed for **C4** is applied for semantic matching on selected directories (i.e. selected SONs). A self-organizing process based on an autonomous super-peer selection algorithm is applied for super-peer SONs construction and maintenance. The system performance is evaluated by simulations and the results indicate efficient service discovery (in terms of recall, messages-per-request and hops-per-request) and efficient SON management (in terms of self-organization time, management procedure overhead and load factor).

Research topic T3 Service instantiation:

- ***C6:*** *Manuscript execution support – State Machine Interpreter (SMI).* This contribution addresses **P3**. This thesis implements an execution support for service instantiation, namely the SMI, which can interpret and execute EFSM-based XML manuscripts. SMI can instantiate the manuscripts according to available capability and status information.

Research topic T4 Service adaptation:

- ***C7:*** *Physical service adaptation.* This contribution addresses **P4.** An approach for physical service adaptation is proposed based on the XML manuscripts. Given a service adaptation request, the system dynamically selects and instantiates XML manuscripts according to runtime capability and status information. The actual execution codes for the behaviour patterns defined in the manuscripts can be dynamically selected according to available capability and status. The dynamic generation of such adaptation requests according to traffic situation and failure states is not considered.
- ***C8:*** *Dynamic service management framework.* This contribution addresses **P4** and is related with **C2**, **C6** and **C7**. A RM-based framework integrating service behaviour specification (i.e. EFSM-based XML manuscript), selection (instantiation) and adaptation is proposed and prototyped. Selection and Mapping Rules are proposed and modelled.

For research topics T1-T4:

- C9: *Prototypes and simulations.* This contribution addresses **P5** and is used to evaluate and validate the proposed frameworks and mechanisms.

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) for partial fulfilment of the requirements for the degree of *Doktor Ingeniør* (Dr. Ing). The work presented in this thesis has been carried out at the Department of Telematics, NTNU, Trondheim, under the supervision of Professor Finn Arve Aagesen. The doctoral work has been financed by the Norwegian Research Council for three years and by the Department of Telematics, NTNU for one year.

This thesis addresses some issues related to service engineering and service management in adaptable service systems. The thesis is composed of three parts:

- *Part I*: *Introduction.* It describes the background of this research and gives an overview of Part II of the thesis.
- *Part II: Included papers*. It is a collection of seven papers (PAPER A to PAPER G) published at different conferences and workshops. Each paper deals with one or several of the research topics, namely, *service representation*, *service discovery*, *service instantiation* and *service adaptation*.
- *Part III: Appendices*. Additional information related to super-peer SON service discovery system is provided. These appendices help understand PAPER B and PAPER C and are thus integral part for service discovery.

Acknowledgements

Many people have helped me through the course for the pursuing of my doctoral degree. First of all, I would like to give my sincere thanks to my supervisor, Finn Arve Aagesen. Without his guidance and help, the completion of the thesis would have been a mission impossible. I benefited from his rigorous style of work and his strict requirements on the quality of research. His inspirations through discussions and valuable feedbacks made the journey of research more effective and enjoyable. The art of research and the skills of writing scientific papers have also been accumulated through the journey.

I am grateful to all colleagues at the Department of Telematics and all the members of the TAPAS project. In particular, I would like to thank Professor Ole Petter Håkonsen for four years' harmonic cooperation when I worked as a teaching assistant for his course "ICT and Market". His understanding and consideration as well as the impressive lectures have made our cooperation a pleasant experience. I am also grateful to Professor Steinar Andresen, who is the first guide for my research career at this department. Special thanks go to Randi Flønes for all her administrative support and advices. Thanks to Mazen Malek Shiaa for cooperation on the papers and valuable discussions on TAPAS project. Thanks to Cyril Carrez and Hao Ding for constructive discussions during the co-authoring of papers. My sincere thanks also go to Steinar, Mazen and Cyril for their patience in thesis reading and the valuable feedbacks. Their comments have assisted me greatly in improving the quality of this thesis. Thanks to Paramai Supadulchai and Chutiporn Anutariya for the help with XET engine. Thanks to Jarle Kotsbak, Pål Sæther, Asbjørn Karstensen for technical support. Thanks to Richard Sanders for discussions and the happy time sharing the office and to Jacqueline Floch for the discussions during my early study.

Thanks to my parents in China for their unselfish love and support for all the years. Last, but not least, my sincere thanks to my dearest husband Naiquan, for his continuous support and love, and to our lovely children, Kristian, Fredrik and Astrid, for all their smiles that can dismiss all the troubles and tiredness.

Contents

List of Papers

Table 1 lists papers published that constitute part II of this thesis. Table 2 lists additional papers published as a part of my doctoral work, but not included in this thesis.

Table 1 - An overview of the papers included in part II of this thesis.

PAPER A	Shanshan Jiang and Finn Arve Aagesen. *An Approach to Integrated Semantic Service Discovery.* In Proceedings of Autonomic Networking (AN'06), Paris, France, September 27-29, 2006. Lecture Notes in Computer Science (LNCS) 4195, pp. 159-171, 2006.
PAPER B	Shanshan Jiang, Finn Arve Aagesen and Hao Ding. *A Self-organizing Service Discovery System Based on Semantic Overlay Networks.* Journal of System and Information Sciences Notes, July 2007, Volume 1, Number 3, pp. 303-309. SIWN International Conference on Complex Open Distributed Systems (CODS'07), Chengdu, China, July 22-24, 2007.
PAPER C	Shanshan Jiang and Finn Arve Aagesen. *Efficient Service Discovery System Based on Semantic Overlay Networks.* In Proceedings of 6th International Information and Telecommunication Technologies Symposium (I2TS'07), Brasilia, DF, Brazil, December 12-14, 2007.
PAPER D	Shanshan Jiang and Finn Arve Aagesen. *XML-based Dynamic Service Behaviour Representation.* In Proceedings of Norsk informatikkonferanse (NIK'03), Oslo, Norway, November 24-26, 2003.
PAPER E	Shanshan Jiang, Cyril Carrez and Finn Arve Aagesen. *Automatic Translation of Service Specification to a Behavioural Type Language for Dynamic Service Verification.* In Proceedings of RISE 2004 on Rapid Integration of Software Engineering techniques, Luxembourg, November 26, 2004. Lecture Notes in Computer Science (LNCS) 3475, pp. 34-44, 2005.
PAPER F	Shanshan Jiang, Mazen Malek Shiaa and Finn Arve Aagesen. *An Approach for Dynamic Service Management.* In Proceedings of IFIP WG 6.3 Workshop and EUNICE 2004 on "Advances in fixed and mobile networks", Tampere, Finland, June 14-16, 2004.
PAPER G	Mazen Malek Shiaa, Shanshan Jiang, Paramai Supadulchai and Joan J. Vila-Armenegol. *An XML-Based Framework for Dynamic Service Management.* In Proceedings of IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM'04), Bangkok, Thailand, November 23-26, 2004. Lecture Notes in Computer Science (LNCS) 3283, pp. 273-280, 2004.

Table 2 - An overview of additional papers published as a part of my doctoral work, but not included in this thesis.

[1]	Shanshan Jiang and Finn Arve Aagesen. *Service Discovery Based on Semantic Overlay Networks.* In Proceedings of 3rd Balkan Conference in Informatics, Sofia, Bulgaria, September 27-29, 2007.
[2]	Shanshan Jiang and Finn Arve Aagesen. *Design and Implementation for XML-based Dynamic Service Behaviour Representation.* Plug-and-Play Technical Report, Department of Telematics, NTNU, ISSN 1500-3868, September 2003.

List of Figures

List of Tables

Abbreviations

DHT	Distributed Hash Table
DPE	Distributed Processing Environment
EFSM	Extended Finite State Machine
IN	Intelligent Network
IP	Internet Protocol
ISO	International Standards Organization
ISO/OSI RM	ISO Reference Model for Open System Interconnection
NGN	Next Generation Network
ODP	Open Distributed Processing
P2P	Peer-to-Peer
QoS	Quality of Service
RM	Reasoning Machine
SCP	Service Control Point
SDP	Service Data Point
SIB	Service Independent Building Block
SON	Semantic Overlay Network
SMI	State Machine Interpreter
TAPAS	Telematics Architecture for Play-based Adaptable Service Systems
TINA	Telecommunications Information Networking Architecture
XML	eXtensible Markup Language

PART I: INTRODUCTION

Part I Introduction

This introduction part gives service related definitions used in this thesis and discusses the key technologies for networked services, in particular, adaptable service systems and peer-to-peer technology. It presents the research objectives, problem statements and scope for this research. The problem statements are further mapped into four research topics. As the basis and context for this research, the related concepts and architectures in TAPAS are introduced. This thesis's contribution to TAPAS is also described. The research topics covered in the thesis as well as research contributions within each topic are presented. The research methodology and research cycle for this thesis is described. This part also summarizes each of the included papers in Part II and their research contributions. Summary of the results, conclusions and future work are also included.

1. Service Related Definitions

This section presents service related definitions used in this thesis. Section 1.1 introduces networked services and gives examples of user services. Section 1.2 presents service model concepts in general as well as service models to be used in this thesis. Section 1.3 presents service life cycle concepts, while Section 1.4 describes the service ontology defined for this research.

1.1 Networked Services

A *service* is a functionality offered to a service user by a service provider. Both service users and service providers can be human beings, enterprises, as well as software and hardware entities (i.e. programs and devices). A *networked service* is a service provided by a *service system* through the structural and behavioural arrangements of *service components* based on a communication infrastructure. A service component is an entity that offers some functionality. A service system is constituted by service components, which in turn can also be service systems, thus constituting a hierarchy of service systems. Such relationship is illustrated in Figure 1. This figure is a subset of a more complete service concept model defined in Figure 3.

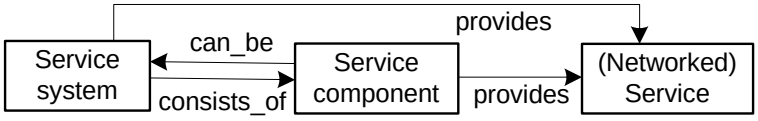


Figure 1 - Service system, service component and (networked) service.

A service provided to a human user is usually referred to as a *user service*. These days the amount and variety of user services are growing dramatically, due to the radical advances of technology, especially the development and use of Internet for e-business, e-banking and other types of networked services provided by private and government service-oriented enterprises. To get a feeling of the versatility of user services, Table 3 lists 9 user service areas with examples of service functionalities. This table is a slightly different version of the service model presented in [RPOS05], which

classifies services into 8 service areas. The main difference is that we think *information retrieval/transfer* should be a separate user service area.

Table 3 - User service areas and examples of service functionalities.

User Service Areas	Examples of service functionalities
Broadcasting/weather/public	Pay broadcasting, emergency news, electronic voting
Business/commerce/banking	Management, reservation, membership, payment, advertisement, brokerage
Communication	Voice/moving picture communication (e.g. telephony, video conference), text communication (e.g. email, chat, instant messaging), roaming
Emergency/disaster	ITS(camera, traffic signalling, alarm), satellite, mobile control centre, emergency routing and traffic handling schemes
Information retrieval/transfer	Web browsing, computational services, data/file transmission
Leisure/games	Multimedia quality, contents selection, online gaming, television, Video on Demand, smart home
Life	Intelligent agent, personal information management, monitoring, remote education, access and security control
Mobility/traffic	Navigation, context-aware, position information, lost protection, positioning-based guidance
Telemedicine/health	Remote control services, remote medical consult and treatment

1.2 Service Models

A service model has concepts defined by the type of abstractions and life cycle phases of the service. The abstractions mechanisms are viewing, scoping and layering. The viewing defines which aspect and the scoping defines the level of detail. Layering is the hierarchical arrangement of functionality which can apply both viewing and scoping. A viewpoint provides an abstraction of the service from a certain perspective, while layering models the service using abstraction layers with order and hierarchy. This Section 1.2 will present abstraction model. Life cycle models are presented in Section 1.3. The interaction of the components constituting an executing service requires concepts defined by a common ontology. Ontology concepts are defined in Section 1.4.

Viewpoints and layering principles for networked service design and interoperation were introduced by *ISO Reference Model for Open System Interconnection* (*ISO/OSI RM*) [DZ83] and *ISO Reference Model for Open Distributed Processing (ODP)* [ODP95]. ISO/OSI RM defines a seven-layer model, which consists of *physical, data link, network, transport, session, presentation* and *application* layers. Within the context of ISO/OSI RM, a (N)-service is the service offered to the (N+1)-layer by the (N)-layer

using the (N-1)-services. With reference to this model, networked application services based on and/or residing at the application layer are considered in this thesis. From now on, such networked application services are referred to as networked services, or simply services.

The ISO/OSI ODP defines a viewpoint model, which consists of *enterprise, information, computational, engineering* and *technology* viewpoints. The enterprise viewpoint specifies the roles of the external actors of the system and their relationships. The information viewpoint describes the semantics of the information and information processing. The computational viewpoint describes the functional decomposition of the system into objects that interact at interfaces. The engineering viewpoint specifies the infrastructure required to support distribution. The technology viewpoint gives the system implementation of hardware and software components. These viewpoints are widely used in the design of distributed systems. TINA [BDD99] (see also Section 2.1) introduced a specific enterprise viewpoint model denoted as the business model. This model defines the relationship between the stakeholders related to service delivery, usage and business. TINA also introduced a specific two-dimensional architecture of the service:

- The *computing architecture* is a generic model for the modelling of any service system.
- The *service architecture* is the structure of services and service functionality components.

While the computing architecture focuses on the modelling of functionality with respect to implementation, but independent of the nature of the service functionality, the service architecture has focus on the structural arrangement of service functionality independent of implementation.

Considering the computing architecture, this thesis considers the service at three abstraction levels, denoted as *conceptual, engineered* and *physical* service [Aag07] defined as follows:

- *Conceptual service*: the high level abstraction of the service independent of how it is realized.
- *Engineered service*: the system and procedures for the realization of the service based on concepts that can be executed and implemented.
- *Physical service*: the real provision of the service by instantiated software and hardware components in nodes as well as supporting communication infrastructure.

This generic *three-level service model* is illustrated in Figure 2. Each service model is constituted by *conceptual service components*, *engineering service components* and *physical service components,* respectively. A *conceptual service component* is an abstract high-level component that constitutes a conceptual service. An *engineering service component* is a component suitable to map the conceptual service components to programming languages. A *physical service component* is an operating system process component that executes in a node. The conceptual/engineered/physical service is

realized by the structural and behavioural arrangement of conceptual/engineering/physical service components, respectively.

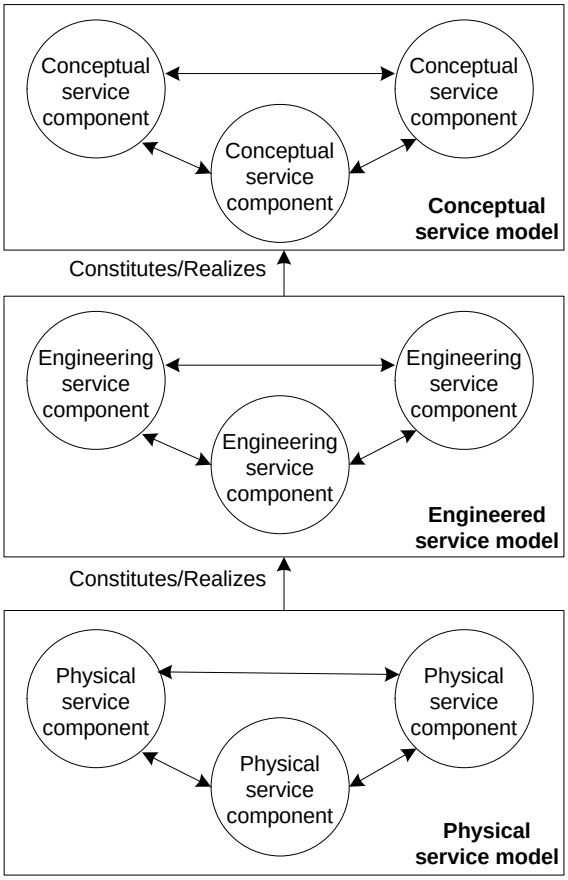


Figure 2 – A three-level service model [Aag07].

Figure 3 presents important service model concepts without paying attention to the level of abstraction.

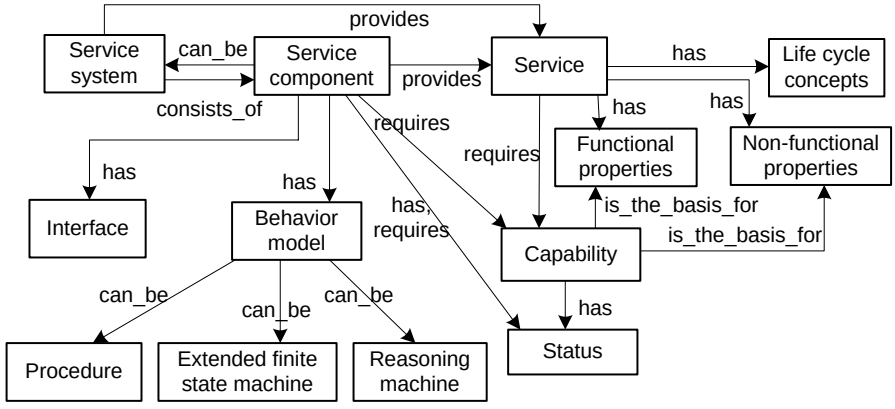


Figure 3 - A general service concept model.

The concepts of *service*, *service system* and *service component* have already been introduced in Section 1.1. A service has *functional properties* and *non-functional properties*. *Functional properties* of a service are usually referred to as its *functionality*. A service model needs concepts related to the life cycle phases. The various *service life cycle concepts* will be discussed in Section 1.3. Functional and non-functional properties will be discussed in more detail in Section 1.4.

A *capability* is an inherent property of a node, which is a basis for implementing a service [AS07]. Capabilities can be classified as functions, resources and data [ASAS05]. Examples of capabilities are software programs, library functions, CPU, memory, transmission channels, hard disk, user profiles and access rights. *Status* is the measure for the situation in a system with respect to the actual number of nodes, active entities, traffic situation and Quality of Service (QoS) [ASAS05]. Status can both comprise observable counting measures, measures for QoS or calculated predicates related to these counts and calculated measures.

A service requires certain capabilities, which are the basis for the functional and non-functional properties of the service. A service component has certain status, and at the same time, may require certain capabilities and status for the provision of a service. The concepts of capability and status provide the flexibility to map the concepts in the conceptual service model to computing and communication platforms in the physical service model.

A service component has a *behaviour model* and an *interface*. Based on the nature of the service, a *behaviour model* can be:

- *Procedure*. A procedure is basically a state-less *operation* that has *preconditions* and effects, and may generate *outputs* according to given *inputs*.
- *Extended Finite State Machine (EFSM)*. A *state machine* is a model of behaviour composed of a number of *states*, *transitions* between those states, and *actions*. An EFSM is a state machine with finite states and variables, which performs actions (including calculation and updating of variables), sends outputs and moves to a next state when receiving messages.
- *Reasoning Machine (RM)*. A RM is an information processing system (e.g. a computer program) that can derive a conclusion by systematically employing inference steps, which process the rules, cases, objects or other types of knowledge and expertise based on the facts of a given situation.

A service component *interface* can be:

- A user-component interface: the interface between a service user and a service component.
- A component-component interface: the interface between two service components.

1.3 Service Life Cycle Concepts

The main *service life cycle concepts* can be briefly distinguished into two groups, i.e., concepts related to *service engineering* and concepts related to *service management*, as illustrated in Figure 4.

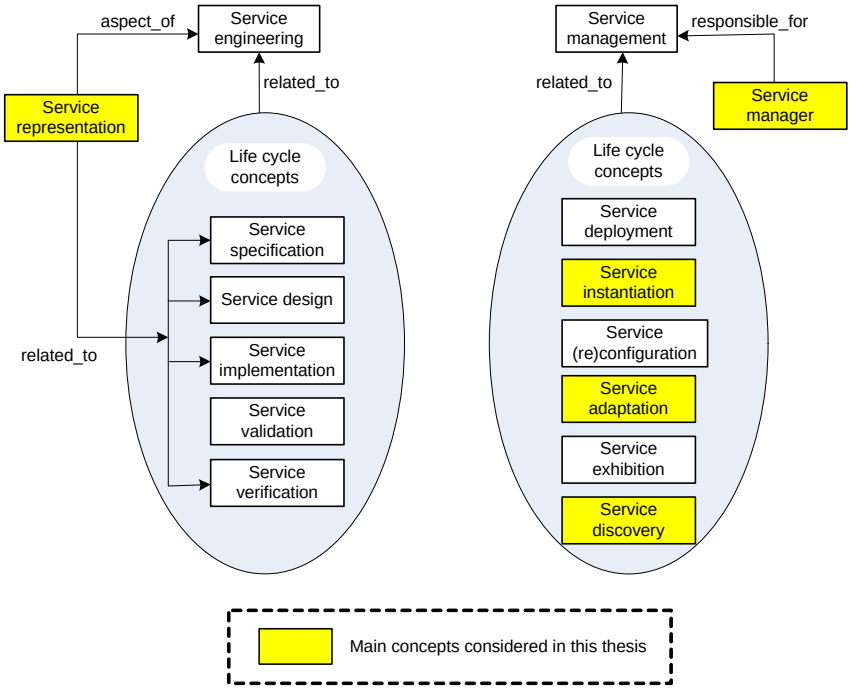


Figure 4 - Concepts related to service engineering, management and life cycle.

Service engineering refers to the creation of conceptual, engineered and physical services. The main service life cycle concepts related to service engineering are:

- *Service specification* is the process that creates the conceptual service according to the requirements.
- *Service design* is the process that creates the engineered service.
- *Service implementation* is the process of producing the physical service.
- *Service validation* is the process to confirm that the service produced (conceptual, engineered or physical service) satisfies its requirements.
- *Service verification* is the process of checking service specifications to ensure that service components can play well together.

In addition, one important aspect of service engineering is:

- *Service representation,* which is the representation of a service (conceptual, engineered and physical) based on a specific language and a data model. It is thus related to several life cycle concepts, i.e., service specification, design, implementation and verification.

Service management is the functionality to control the provision of service functionality and quality of a service, both within a service system and across different service systems, through the service life cycle phases. The main service life cycle concepts related to service management include:

- *Service deployment* introduces new services and service components into the physical nodes to make them ready for use.
- *Service instantiation* is the process of creating a service instance upon request and making it available to the user. This is the provision of the physical service.
- *Service configuration* is the arrangement and setup of service systems. It can be done initially during service deployment and instantiation, or later during service execution, which is also called *service reconfiguration.*
- *Service adaptation* is the process of adapting the structure or behaviour of the service to the changes during its execution.
- *Service exhibition* refers to the exposure of functionality and other service related properties to service users, so that the service can be discovered and executed. It is also called *service publication* or *service advertisement.*
- *Service discovery* is the process of finding the services that satisfy the functional and non-functional requirements. Depending on the service request, the result can be the conceptual or engineered service (service type), or the physical service (service instance). Service discovery can also be applied to service components in order to compose services, both statically (during service design) or dynamically upon a request.

Service manager is responsible for service management. It is an entity that carries out service management functionality, such as the functionality required for service adaptation, discovery and instantiation.

1.4 Service Ontology

Ontologies are the basis for adding semantic expressiveness to service descriptions and requirements. An *ontology* is an explicit and formal specification of a shared conceptualization [SBF98]. A *service ontology* is accordingly an explicit and formal specification of core concepts of the functional and non-functional properties of service. An *upper ontology of service* is a model of common service-related concepts applicable to a wide range of domains.

Ontology provides several advantages due to its semantic expressiveness. On one hand, service ontology enables semantic interoperability as shared concepts with common semantics can be defined. On the other hand, ontology facilitates automatic reasoning because the meaning of the concepts can be accurately and automatically interpreted. The ontological relationships, such as subClassOf, enable comparison of semantic similarity of the concepts defined in the ontology.

In order to further represent functional and non-functional properties with semantic expressiveness, an upper ontology of service[1] is defined for the conceptual service, as shown in Figure 5. The conceptual service functionality is based on procedures, and represented using *operations, inputs, outputs, preconditions* and *effects*. *Non-functional properties* include *service category, service parameter*, *QoS parameters* and *policies*. *Service category* is a categorization of service. The set of service categories is connected by a rooted tree, *category hierarchy (CH)*. *QoS parameters* are QoS attributes that can be expressed in quantifiable measurements or metrics, such as reliability, availability, security and performance measures. *Service parameters* are other service related parameters, such as price and location. *Policies* are the constraints applied on the system. Policies can be *business policies*, *QoS policies* and *context policies*. *Business policies* are rules related to business concepts, such as a delivery policy. *QoS policies* are rules related to QoS parameters. *Context policies* are rules related to context information. Examples of context information are location, time, connection, user's feeling, presence, and user's habits and hobbies.

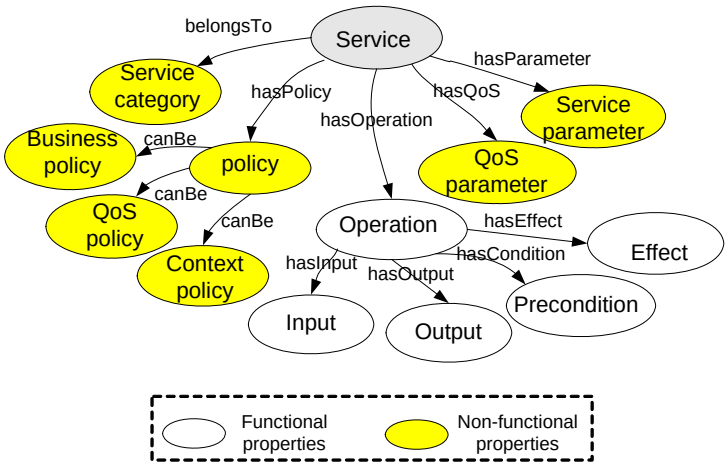


Figure 5 - Upper ontology of service.

[1] This ontology was first defined in PAPER A, in order to represent the conceptual service. It was extended with the concept of *service category* in PAPER B and PAPER C in order to organize directories into Semantic Overlay Networks (SON). It is sometimes in this thesis referred to as simply the *service ontology*.

2. Key Technologies for Networked Services

2.1 Overview

Networked services have been an important research topic since the introduction of the computer controlled telephone exchanges in the early 60s. The research focus on networked services during the 80s and 90s was primarily on *service architectures* that give flexibility and efficiency in the definition, design, deployment and execution of the service. *IN* (*Intelligent Network*) and *TINA* (*Telecommunications Information Networking Architecture*) are such examples.

IN [IN92] was introduced in the 1980s as a promising way to defining and constructing services to cope with the complexity in services. Its main purpose was to increase flexibility in service design and speed up the provision of new services by implementing new services from predefined service-related functionality components, called *SIBs* (*Service Independent Building Blocks*). IN architecture separates service control from switching and places service logic in dedicated network components (i.e. *service control points*, or *SCPs*). Database has been an important component in IN architecture and is used to realize various aspects of flexibility. SCP contains programmable service logic and utilizes such databases either internally or by consulting external *service data points (SDPs)*. The IN architecture, however, has limitations for advanced and complex service provision as the execution of the IN services can only be triggered from switches.

TINA [BDD99] was developed between 1995 and 1997. Its primary objective was to become a software architecture for services and for the operation of these services, by putting together the best of telecommunications and information technologies [BDD99]. TINA provides concepts and principles for the design, deployment, operation and management of the TINA-based services based on a notion of sessions, and uses *DPE* (*Distributed Processing Environment*) to resolve heterogeneity and distribution. TINA offers greater flexibility than IN as it separates information network from transport network and offers an abstract view of resources to control and manage.

Since the middle of the 1990s, Internet usage has undergone exponential growth and various initiatives to increase the power of flexibility within Internet have emerged. Examples include *Active and programmable networks*, *Web Services* and *Semantic Web*.

Active and programmable networks [TSS97][CDK99] are aimed at speeding up the efficiency of application layer services. Such networks give more intelligence to routers which can adjust themselves to the environment using network-level programming capabilities. The intermediate routers have functionality up to the application layer and can carry out customized operations on the packets. In addition, users can program the network by injecting their programs into it.

Web Services technology [WS07] deals mainly with service descriptions, service discovery and utilization. It provides high level interfaces, focusing mainly on service *syntax*, for service interoperability over Internet. On the other hand, *Semantic Web* [SW07] is a semantic description framework for *semantic* interoperability, enabling the encoding of knowledge and services on the Web using ontology-based semantics. Semantic Web technologies will not only provide essential tools for semantic representation but also facilitate automatic reasoning and advanced decision-making.

Through more than one century's development, telecom networks have evolved as the world's largest online, real-time distributed system and offer world-wide connectivity with vast amount of various networked services. *Next Generation Network (NGN)* is a general term referring to the future architectural evolutions in telecom networks. NGNs are generally packet-based, IP-based, multi-service networks [NGN04]. The fundamental difference between NGNs and current telecom networks is the transition from *circuit-switched* networks to *packet-based* networks such as those using *Internet Protocol (IP)*. NGN provides a unified and flexible control environment that allows unrestricted access technologies to use the same core network and supports multimedia services with generalized mobility. One of the major characteristics of NGN is the distribution of network intelligence [NGN07], i.e., intelligence (control) is distributed throughout the network and may reside at the edge or in the network as needed.

Due to the increasing complexity and heterogeneity in today's distributed computing, networking and service systems, the research focus on networked services has recently shifted towards *adaptability* and *evolution* of services. Examples are *adaptable service systems* and *autonomic systems.*

Adaptable Service Systems are service systems that are able to adapt dynamically to changes in time and position related to users, nodes, capabilities, status, changed service requirements and policies [ASAS05]. Adaptable Service Systems will be presented more comprehensively in Section 2.2.

Autonomic Systems are service systems that can act upon the failures, unexpected events and changes in environments and requirements in an autonomous and adaptive manner. They are targeted at coping with the rapidly growing complexity of operating, managing and integrating service systems, and also towards the management of network and computing resources in a decentralized manner. Different approaches have been proposed: autonomic computing [AC07], autonomic communication [ECAC07] and autonomic networking [ANA07][Bion07], which are attempts to create more self-managing and self-organizing systems from computing, communication and networking fields, respectively. Basically, *robustness, adaptability, intelligence* and *dependability* are among the goals desired for autonomic systems [DDF06].

Concerning the network infrastructure for networked service systems, there are basically two types of architecture: *client-server* based and *peer-to-peer* (*P2P*) based. In client-server based architecture, a node has predefined role as either a client or a server, with clients making requests to dedicated servers. In P2P-based architecture, each node is considered equal (thus the name *peer*), and can act both as a client or a server to other peers. The client-server based architecture may suffer from single-point-of-failure and scalability problems due to dedicated servers. In addition, some servers may have performance bottleneck due to sharply increased requests, while others may have plenty of unused resources. P2P-based architecture holds many promises for robustness and scalability and can alleviate the above problems facing the client-server architecture. Section 2.3 will give a more detailed description of related P2P technology.

2.2 Adaptable Service Systems

In a dynamic environment, components come and go all the time, and the services will be provided by nodes and devices that experience great variation in their available capabilities. Therefore, a service system needs to adapt or evolve due to changes in

human needs, enabling technology or application environment during its life cycle. Some of the changes can be predicted at the design time, while others are not. This leads to a need for dynamic adaptation or evolution, i.e. modification or extension of the system without interrupting those parts of the system that are not directly affected [KM90].

As defined in Section 2.1, *Adaptable Service Systems* are service systems that can adapt dynamically to the various changes. *TAPAS* (*Telematics Architecture for Play-based Adaptable Service Systems*) project [ASAS05] is such an example research project. The defined adaptability functionality for adaptable service system is very broad. In fact, according to this definition, adaptable service systems are more general than autonomic systems, as autonomic systems impose more strict requirements for the systems and environment. Adaptable service systems should possess certain adaptability properties. TAPAS has defined two classes of targeted properties:

- *general adaptability properties and*
- *core functional adaptability properties*

The general properties are properties of the architectural framework, while the core properties are properties of the functionalities.

The *general properties* are in [AS07] defined as follows: 1) There must be a flexible and common way of modelling services, 2) The framework must be flexible with respect to the adding of adaptability properties and features, 3) The service concepts must be flexible and powerful, 4) The software mechanisms must be flexible and powerful, and 5) There must be an easy mapping of service models to software models.

The *core properties* as defined in [ASAS05] and [AS07] are grouped into three classes:

A1: Rearrangement flexibility
A2: Robustness and survivability
A3: QoS awareness and resource control

Rearrangement flexibility means that the system structure and the functionality are not fixed. Nodes, users, resources, services and service components can be added, moved, removed according to the needs. New nodes and capabilities are found automatically when introduced and such information is propagated accordingly. Furthermore, there is a continuous adaptation to changed environments and operation strategies/policies.

Robustness and survivability means that the architecture is dependable and distributed. Resources and functionality are replicated, and malicious and unauthorized components need to be inhibited. It also means that the system can reconfigure itself in the presence of failures and can provide continuous operation through re-initialization.

QoS awareness and resource control means that there is functionality for negotiation about QoS and optimum resource allocation, as well as monitoring of resource utilization and reallocation of resources.

Traditionally, services are defined only at design time. A change in the system usually requires manual modification of service specification, then manually reconfiguration and deployment into the system. Such situation has been changing with

adaptable service systems, where services can be dynamically created, i.e., the service specifications can be changed or created dynamically and deployed during the execution phase.

There are basically two types of service adaptations considering the conceptual and physical service level:

- Conceptual service adaptation
- Physical service adaptation

Conceptual service adaptation is adaptation at the conceptual service level. Service specifications can be dynamically created by composing new service specifications from existing service specifications during the execution phase. For example, when a user sends a request for a certain service, sometimes a service that can satisfy all the user's requirements does not exist. Instead, several services together can provide the required service. The dynamic composition of conceptual services is thus an approach for service adaptation at the conceptual service level.

Physical service adaptation is adaptation at the physical service level, e.g., by selecting different predefined codes for a service dynamically.

For conceptual service adaptation, new services are created dynamically, and they need to be deployed and instantiated. For physical service adaptation, the services are already deployed in the system.

2.3 Peer-to-Peer Technology

Peer-to-Peer (P2P) technology is a distributed computing technology where each node in the system is considered equal (also called *peer*), instead of client-server relationship. Each peer is considered an autonomous entity, and a P2P network is typically used to connect peers via largely ad hoc connections. An *overlay network* is a network which is built on top of another network. Many P2P networks are overlay networks because they usually run on top of the Internet. Advantages of P2P networks are typically robustness (due to its distribution and autonomous nature), scalability (due to no central manager or controller), autonomy and self-organizing, but the security problem is a major concern. [AS04] provides survey and overview of P2P systems and technologies.

There are several ways to classify P2P architectures and networks. The classification defined in [AS04] is followed here. P2P architectures can be classified as *pure, partial* or *hybrid* according to the degree of centralization. *Purely decentralized* architectures have no central servers to coordinate the peer activities and all peers are equal in the roles. *Partially centralized* architectures have dynamically assigned servers (called *super-nodes* or *super-peers*) that assume a more important role than others. *Hybrid decentralized* architectures have a central server facilitating the interaction between peers, which usually constitutes a single point of failure. On the other hand, P2P architectures can also be classified as *structured* or *unstructured* according to whether there is structure of the links between nodes in the overlay network.

Pure, unstructured P2P networks do not rely on any centralized entities and are well-suited for highly dynamic environments such as ad hoc networks. Examples of pure, unstructured P2P networks include Gnutella [LW00] and FreeHaven [DFM00].

Distributed Hash Table (*DHT*)-based P2P networks are formed by constructing a structured overlay network in which each participating node needs to communicate with only a small fraction of the other nodes. By using DHT-techniques, a set of *keys* are assigned among participating nodes, and can be used to efficiently route messages to the unique owner of any given key. The benefit of using DHT lies mainly in its efficiency, where a message can be routed within O(logN) hops for a network of N nodes. However, such structured P2P systems impose restrictions on data or index placement, and have tight coupling between nodes. Examples of DHT-based infrastructures include Chord [SMK01], CAN [RFHK01], Tapestry [ZKJ01] and Pastry [RD01].

Super-peer based P2P networks [YG03] represent partially centralized architectures. A super-peer acts as a central server to a subset of clients, providing services such as listing connected peers and acting as primary connection nodes. Clients submit queries to corresponding super-peer and receive results from it. Essentially, the connections among super-peers form a pure P2P system, and super-peers are responsible for sending and answering requests on behalf of client peers and themselves. It is important to be noted that such super-peers do not constitute single points of failure, since they are dynamically assigned and, if they fail, the network will automatically take action to replace them with others. Super-peer based systems have some degree of centralized management, which provide the efficiency of centralized network as well as autonomy, reliability and load balancing of distributed network. Example systems include JXTA [Gon01] and Kazaa [Kaz07].

Semantic Overlay Networks (*SONs*) have been proposed as an approach to improve the efficiency and quality of search in unstructured P2P systems. The basic idea is to group together peers that contain semantically similar content, so that at search time, queries can be forwarded to only those peers containing content with high probability of satisfying the query constraints. Hence, communication cost of the query can be reduced, while at the same time, result quality can be increased. Evaluation conducted in [CG02] shows that SONs can significantly improve query performance while at the same time allowing systems to decide what content to put in their computers and to whom to connect.

P2P-based service discovery systems have been proposed as promising solutions providing scalability and autonomy [PSNS03][SMK01]. Since different P2P technologies have different research focus, they are suitable for different applications. Pure P2P-based service discovery systems use flooding to route requests and are suitable for highly dynamic environments. However, flooding tends to increase message overhead in the system. DHT-based P2P systems can provide results fast when the request is based on a single key value. They are efficient for service discovery based on exact matching and keyword-based matching, but are not suitable for complex, ontology-based semantic matching. In addition, there is high maintenance overhead for the DHT index.

P2P technology is important as an enabling technology for networked service systems. The reason to use P2P technology in this research is mainly to enhance the robustness and survivability property (core property **A2**).

3. Research Objectives, Problem Statements, Research Topics and Scope

3.1 Research Objectives

The context for this thesis is service engineering and service management in adaptable service systems. In adaptable service systems, new service specifications or modifications to existing services shall be introduced without interrupting the executing services.

The main research objectives are *to specify, construct, evaluate and validate applicable concepts, models, mechanisms, algorithms and frameworks for service engineering and service management in adaptable service systems.*

3.2 Problem Statements

The following problem and sub-problem statements are defined:

P1: How can services be modelled and represented?

- **P1.1:** How to represent conceptual services?
- **P1.2:** How to represent physical services in a flexible manner so that it is possible to adapt the services to changes dynamically?
- **P1.3:** How to extract the component interface behaviour from the physical service representation so that compositional service verification can be applied?

P2: How can services be discovered efficiently, automatically and accurately?

- **P2.1:** How to ensure automatic and accurate service discovery?
- **P2.2:** How to locate services efficiently in a large-scale service system?

P3: How can services be instantiated dynamically and according to available capabilities and status information?

P4: How can new service specifications or modifications to existing services dynamically be introduced without interrupting the executing services?

P5: How to evaluate and validate the proposed frameworks and mechanisms?

3.3 Research Topics

The problem statements P1-P4 are related to the following four research topics:

T1: Service representation

T2: Service discovery

T3: Service instantiation

T4: Service adaptation

In addition, **P5** is related to all the four topics **T1-T4**. Table 4 depicts the relationship between problem statements and research topics.

Table 4 - The relationship between problem statements and research topics.

Problem Statement	Related Research Topics			
	T1	**T2**	**T3**	**T4**
P1	X			
P2		X		
P3			X	
P4				X
P5	X	X	X	X

3.4 Scope

Concerning the targeted adaptability properties defined in Section 2.2, the main emphasis of this thesis is on the functionality that can support or realize core property **A1**. In addition, some support for core property **A2** is also considered, but it is not the main focus.

Concerning the three service levels defined in Section 1.2, service representation is considered at both the conceptual and the physical service level, service discovery is considered at the conceptual service level, while service instantiation and service adaptation are considered at the physical service level. Service adaptation at the conceptual service level, e.g. by dynamically composition of conceptual services, is not considered.

This PhD work is related with TAPAS. TAPAS has a computing and a management architecture in line with the TINA computing and service architecture (cf. Section 1.2 and Section 4). Referring to the TAPAS computing architecture, this thesis focuses mainly on the representation of the conceptual and the physical services. In addition, support for physical services in the core platform is also considered. Methodologies for service engineering are outside the scope of this thesis.

Referring to the TAPAS management architecture, this thesis focuses mainly on service management component and service managers that carry out service management functionality as well as the Service Definitions in service repository. The main assumptions are 1) capability and status information should be available when needed, 2) the changes in the system and environment can be detected and reflected as service requests sent to service managers for appropriate handling.

4. TAPAS

This PhD research is related to TAPAS project (*TAPAS = Telematics Architecture for Play-based Adaptable Service Systems*), which is a research project founded by the Norwegian Research Council and the Department of Telematics at NTNU. On one hand, TAPAS architectures and concepts are the basis and context for this research. On the other hand, this thesis aims to further develop TAPAS concepts, architectures and platform.

This section presents the most related TAPAS features and concepts first, and then summarizes the contribution from the work reported in this thesis to TAPAS.

4.1 TAPAS Architectures

To meet the general properties defined in Section 2.2, the TINA architecture principle presented in Section 1.2 is followed. The TAPAS architecture is separated into a *computing architecture* and a *management architecture,* which corresponds to the TINA computing and service architecture, respectively. The features, concepts and properties of the computing architecture are the fundament for the creation of services with needed adaptable networking services properties.

The TAPAS Computing Architecture

TAPAS *computing architecture* is founded on a theatre metaphor, where actor, manuscript, role figure and capability are core concepts. *Actors* perform *roles* defined by *manuscripts* according to actors' *capabilities*. An actor playing a role is denoted as a *role figure*.

The computing architecture is illustrated in Figure 6. With respect to Section 1.2, the conceptual service model is denoted as the service view, the engineered service model is denoted as the play view and the physical service model is denoted as the physical view.

In service view, the concepts *service system*, *service components* and *service functionality* have been defined in Section 1. *Service performance* is the QoS (or non-functional) aspects of the service.

The leaf service components that can not be further decomposed are constituted by *role figures* in the play view and are implemented by *actors* in the physical view. A role figure plays a *role*, and has *role sessions* with other role figures. A *director* is an actor with supervisory status with regard to other actors.

In the physical view, a *node* is a physical processing unit that has particular *capabilities and status*. A node is installed with the *core platform*, which is a platform supporting the execution of service functionality based on the computing architecture functionality. Actors are implemented by the core platform as generic software components that are able to download and execute different functionality depending on the need. The functionality of an actor is defined in a *manuscript*. The core platform thus has a *manuscript execution support* and a *communication support*.

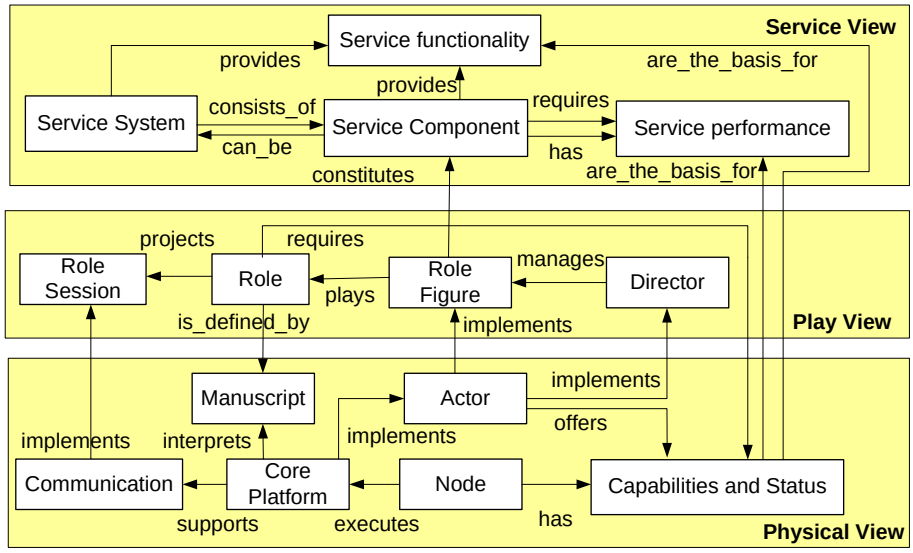


Figure 6 - Simplified TAPAS computing architecture.

The concepts of *capability* and *status* as defined in Section 1.2 represent the attributes of services, service components and nodes. The ability of an actor to play a role depends on the matching of the required capabilities and status of the role and the offered capabilities and status in the node where the actor is executing. This principle is the basis for various service management functionalities, such as the service instantiation and adaptation mechanism proposed in this thesis.

The TAPAS Management Architecture

TAPAS management architecture (Figure 7) defines the functionality structure of and between service systems. *Primary service providing functionalities* comprise the provision of the ordinary services offered to human users. In addition, four main functionality components are defined:

- *Service management*: deployment and invocation of services and service components, selection and adaptation of service specifications as well as service discovery.
- *Capability and status administration*: registration and de-registration of capabilities, as well as updating, transforming and providing access to the capability and status repository.
- *Capability configuration management*: allocation, re-allocation, de-allocation of capabilities, and optimization of the use of capabilities.
- *Mobility management*: the handling of various mobility types.

Two main repositories are defined in the TAPAS management architecture:

- *Inherent capability and status repository*: the snapshots of available capabilities and status information in the system.
- *Service repository*, which consists of:

- *Service configuration definitions,* which include targeted capabilities and status as well as additional requirements when deploying and instantiating a role figure.
- *Service definitions,* which include generic service descriptions (i.e. conceptual service representations) as well as execution definitions (i.e. physical service representations).

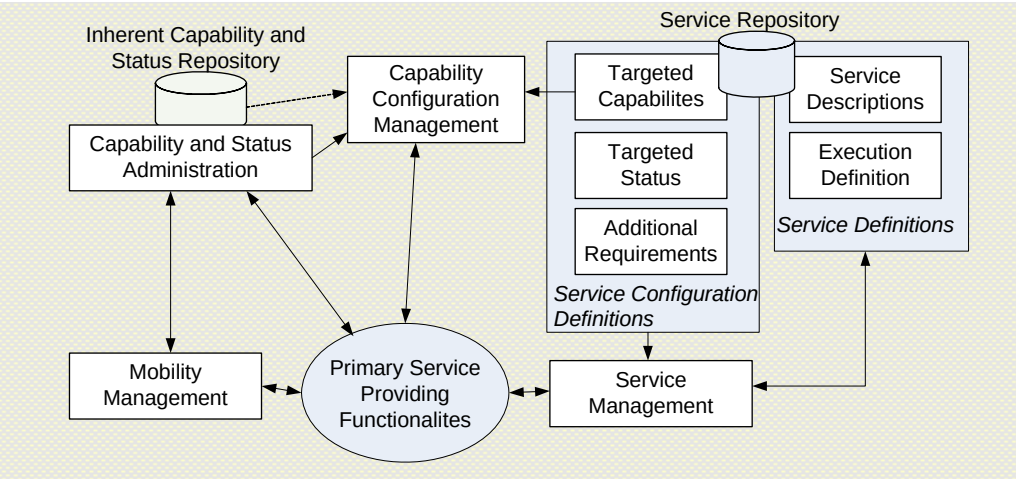


Figure 7 - TAPAS management architecture.

4.2 This Thesis's Contribution to TAPAS

The work in this thesis has contributed to concepts, models, algorithms, mechanisms and frameworks of the TAPAS. Figure 8 illustrates the contributions. At the starting point of this PhD work, the status of TAPAS was as follows:

- The concept model constituting the TAPAS computing architecture was defined [AHAS03].
- The original core platform provided basic communication support and manuscript execution support based on Java RMI and Web technologies [AHAS03].
- The management architecture was originally the configuration management architecture [AASH02], and later functionality components concerning mobility, capability and service management were introduced [ASAS05]. However, no mechanisms and functionalities for service management were defined.

The computing and management architecture as well as the core platform have been extended and revised. In addition, new concepts have been introduced. The following description highlights this thesis's contribution to TAPAS, while detailed description of thesis contribution will be provided in Section 5.

Concerning the computing architecture, this thesis contributes mainly in the service view and the physical view. The TAPAS service view is very generic, applicable to any service model. The work in this thesis defines new concepts, called *upper ontology of service*, for the representation of conceptual services (e.g. QoS parameters, service parameters and policies), as shown in Figure 5 in Section 1. This service ontology model and its representation are described mainly in PAPER A. As for the physical

view, although originally it was specified to model a manuscript as an EFSM [AHAS03], no explicit data model and representation was defined. An EFSM manuscript data model and its XML representation were introduced in PAPER D.

Concerning the TAPAS core platform, extensions and new mechanisms have been proposed by this thesis. In the original version, a manuscript was an EFSM directly represented as a Java object. To achieve greater flexibility and interoperability, XML is introduced as a service behaviour representation language. Action Library and State Machine Interpreter (SMI) were added to the core platform (cf. PAPER D and F), so that XML manuscripts can be easily introduced to the system without recompilation.

Concerning the TAPAS management architecture, originally there was only a basic support for the execution of primary service functionality. This thesis contributes with concepts, models, mechanisms, algorithms and frameworks for service discovery, instantiation and adaptation (cf. Section 5). As for service repository, service descriptions and execution definitions were added. The data model and representation for conceptual service descriptions and execution definitions have been defined.

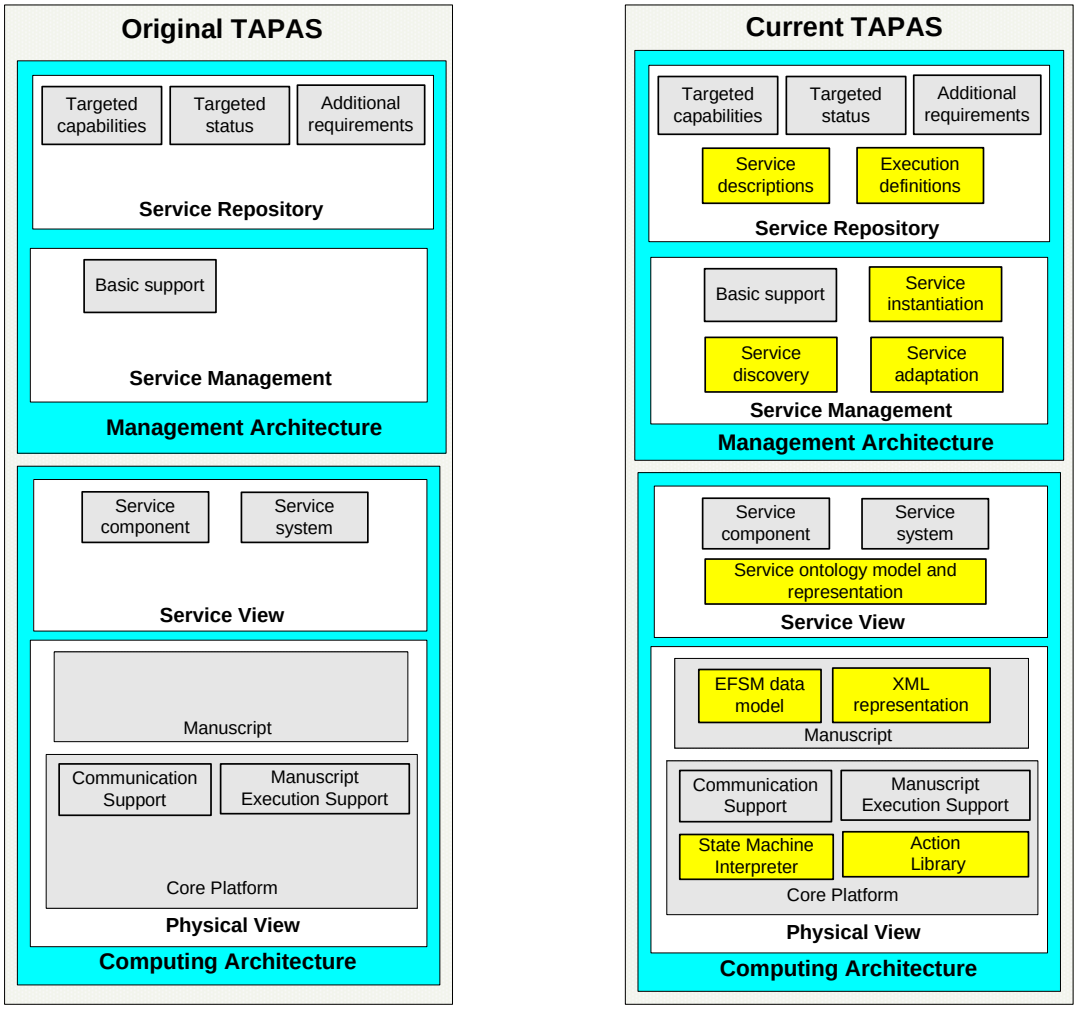


Figure 8 - Contribution to TAPAS architectures.

5. Research Contributions

5.1 General

Problem statements P1-P5 and research topics T1-T4 were defined in Section 3. The substance of the contributions of the research work is presented in the papers included in Part II. The major research contributions are classified as nine contributions **C1** – **C9**, which will be presented in the following subsections.

C1: Conceptual service representation

C2: Physical service representation

C3: Preparation for service verification

C4: Semantic service discovery procedure

C5: Super-peer Semantic Overlay Network (SON)-based service discovery system

C6: Manuscript execution support – State Machine Interpreter (SMI)

C7: Physical service adaptation

C8: Dynamic service management framework

C9: Prototypes and simulations

Figure 9 gives an overview of the research topics and research contributions. It also shows that the topic ***T1*** - *service representation* is the basis of other three topics.

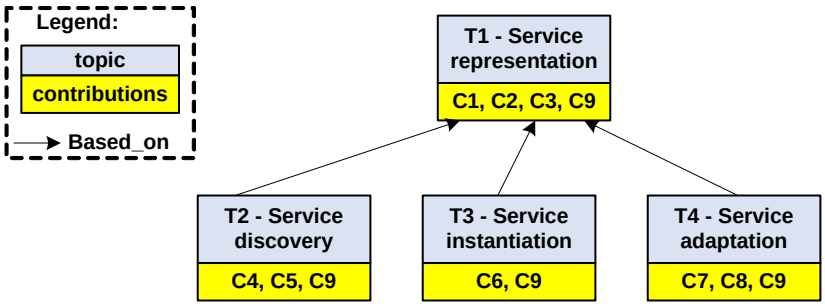
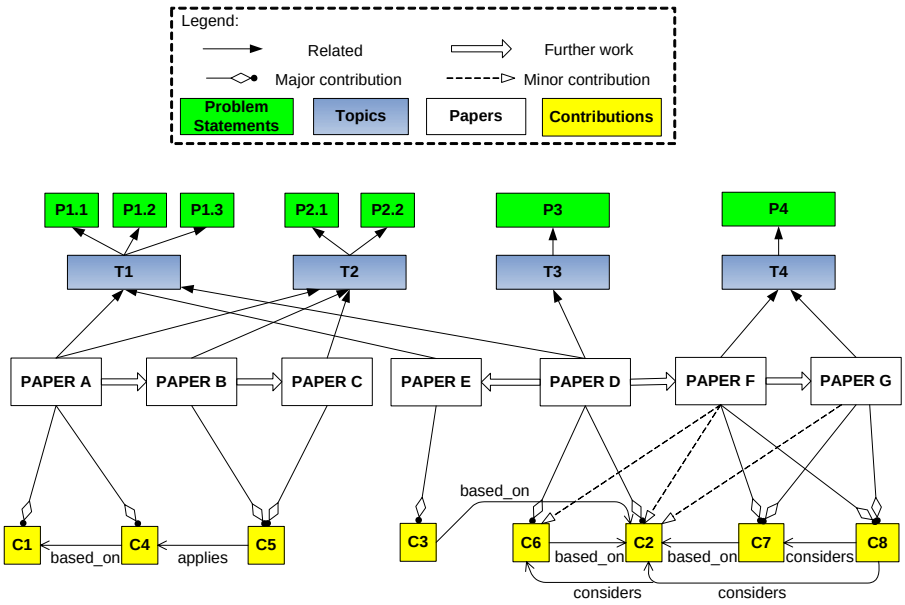


Figure 9 - Overview of research topics and research contributions.

Figure 10 illustrates how the major contributions in the included papers relate to the research topics. Each topic is covered by a number of papers, and each paper constitutes one or several contributions. For instance, PAPER A addresses two contributions **C1** and **C4**, and is related with the topics **T1** and **T2**. The relations between the papers are also illustrated. For example, PAPER C is a further work of PAPER B. In addition, the figure also shows the relationship between contributions. For example, **C4** is based on **C1**. The relationship between problem statements and research topics is also depicted.

The relationship between papers, problem statements and contributions will be further illustrated in Figure 18.



Note: 1. Since **P5** is related to topics **T1**-**T4**, and is addressed by all the papers, **P5** is not depicted in the figure.
2. Since **C9** is related to **P5** and is addressed by all the papers, **C9** is not depicted in the figure.

Figure 10 - Problem statements, research topics, papers and contributions.

The following Sections 5.2-5.5 discuss the various research topics and research contributions. General description of each topic is provided first, and then the main results and major contributions of this thesis within each topic are described. The relationship between the contributions and the problem statements is discussed in Section 5.6. Finally, Section 5.7 presents suggested paper reading guidelines.

5.2 Topic T1: Service Representation

As defined in Section 1.2, a service can be a conceptual service, an engineered service or a physical service. In order to create a service, the service needs to be modelled according to its nature and be represented using proper languages.

Depending on the nature of service, service functionality can be modelled using different behaviour models, such as procedures, EFSMs and reasoning machines, as defined in Section 1.2. These models can also be represented using various languages. The most used languages include SDL, UML, and XML.

SDL [SDL00] is a specification and description language for modelling of behaviour defined by ITU-T. It has formal semantics and has been widely used in the telecom industry. For instance, the work of [Flo03] uses SDL for modelling of service roles.

UML [UML07] is a leading modelling language defined by OMG, originally from the area of object-oriented development. The recent version UML2 [UML05] has included most of the expressive power from other languages, such as SDL. Although it has been attacked as lacking formal semantics, UML has gained a dominating position as a modelling language in the IT community. Work on service modelling based on UML2 can be found in [SB04][SCKB05][CB06].

XML [XML06] is a standard for interchanging structured documents over the Internet. Its main advantages are flexibility and interoperability. Recently a number of languages have been developed based on XML, such as the languages for Web Services and Semantic Web. These languages aim to make the representations both readable for humans and comprehensible by machines. In addition, ontology is used to enhance semantic expressiveness and to facilitate automatic reasoning. XML-based ontology language is thus an important approach to achieve both semantic interoperability and intelligence based on automatic reasoning. Representative efforts on service representation based on XML languages are WSDL [WSDL01] and OWL-S [OWLS03]. The representative XML-based ontology language is OWL [SWM04].

Considering the popularity of XML-based technologies (including ontologies) as well as the advantages for interoperability and autonomic reasoning, this thesis applies XML-based languages for service representation[2]. The research in this thesis focuses on the representation of *conceptual* services and *physical* services using such XML-based languages. In addition, work has been carried out on preparation for service verification.

C1: Conceptual Service Representation

This contribution addresses the problem statement **P1.1**. In this thesis, a high level conceptual service representation based on procedures is proposed. PAPER A presents a model for integrated semantic service description based on a service ontology. The upper ontology of service has been given in Figure 5 in Section 1. The service functionality is represented as operations, inputs, outputs, preconditions and effects. The non-functional properties are described as QoS parameters, service parameters and policies including business policies, QoS policies and context policies. The integrated semantic service description is represented using XML-based languages from Web Services and Semantic Web standards, i.e. WSDL, OWL, and WS-Policy [Baj06]. An example of the integrated semantic service description is given in Figure 11.

Such service description is the basis for semantic matching in service discovery. Service functionality, policies and QoS parameters are all ontology-based. In addition, rule-based policy representation is applied. The application of rules and ontologies enables a rule-based Reasoning Machine (RM) to carry out automatic and semantic matching in service discovery (cf. 5.3).

C2: Physical Service Representation

This contribution addresses the problem statement **P1.2**. Manuscripts define the physical service functionality. PAPER D presents an approach for XML-based EFSM service behaviour representation, and the data model for such XML manuscripts. This model has been extended in PAPER F and PAPER G. Such XML manuscript representation is the basis for service instantiation and adaptation.

[2] This thesis focuses on service representation. Methodologies for service creation from requirements to implementations are out of the scope of this work. In fact, any service engineering methodology or tools, e.g., UML-based techniques, should be applicable for service creation from requirements to implementations. Such created models and specifications can then be translated to our XML-based representations to enable the flexibility and adaptability functionality in later service life cycle phases.

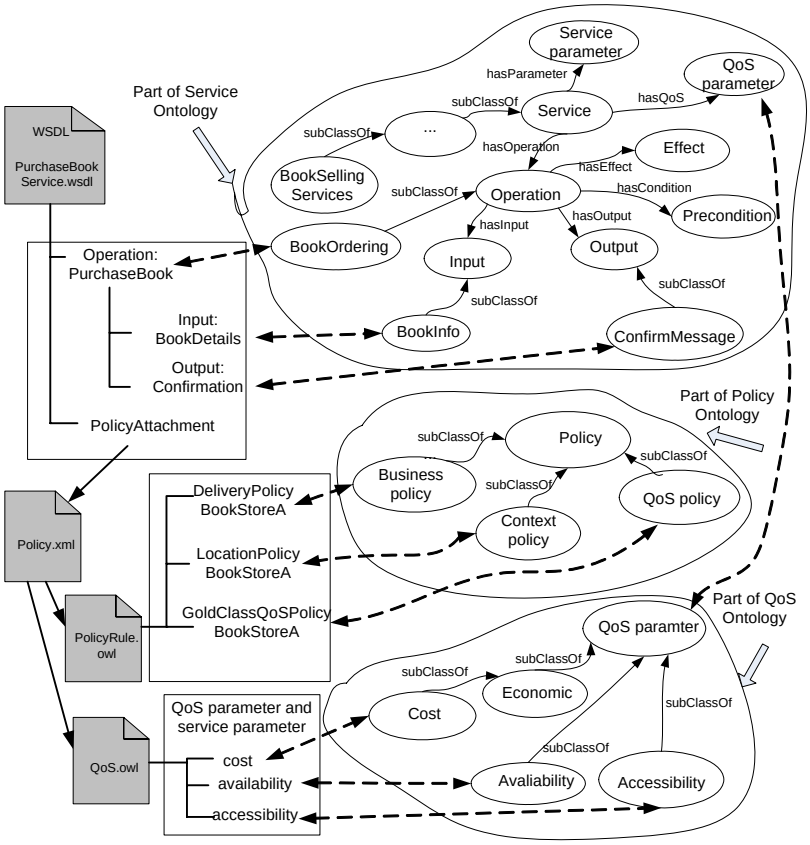


Figure 11 - The integrated semantic service description (cf. PAPER A).

The main advantage of using XML as a service behaviour representation is interoperability and flexibility. An XML manuscript can be directly downloaded to any node and can be executed with the support of State Machine Interpreter (SMI, cf. Section 5.4), without the need of recompilation. In addition, using XML for service behaviour (execution) representation gives one integrated representation of all dynamic service related functionality in TAPAS.

To use XML for the representation of service logic poses a challenge, since XML is inherently a language for structure and data. This problem is addressed by utilizing node inherent capabilities and making the XML manuscript platform independent. The behaviour specification has been separated into *Manuscripts* and the *Action Library*, so that behaviour can be specified in platform independent (generic behaviour patterns) and platform-dependent parts (which can be extended or adapted according to the available capabilities and status information). Figure 12 gives the EFSM-based XML manuscript data model. The behaviour description defined in the manuscript consists of states, data and variables, inputs, outputs and different actions. The <Action> list in manuscripts only specifies the action types, i.e., the method name and parameters for the action. The action types are classified into *Action Groups* according to the nature of actions. The XML manuscripts therefore specify parameterized behaviour patterns. The actual implementation of actions is realized by codes for different platform contained in the Action Library. The action codes in the Action Library are classified into *Capability Categories* according to the dependability on capability. A mapping (called *Mapping table*) is required to link the action definitions in the service specification to the

executable codes stored in Action Library. The selection, instantiation and adaptation of the defined XML manuscripts will be further described in Section 5.4 and Section 5.5.

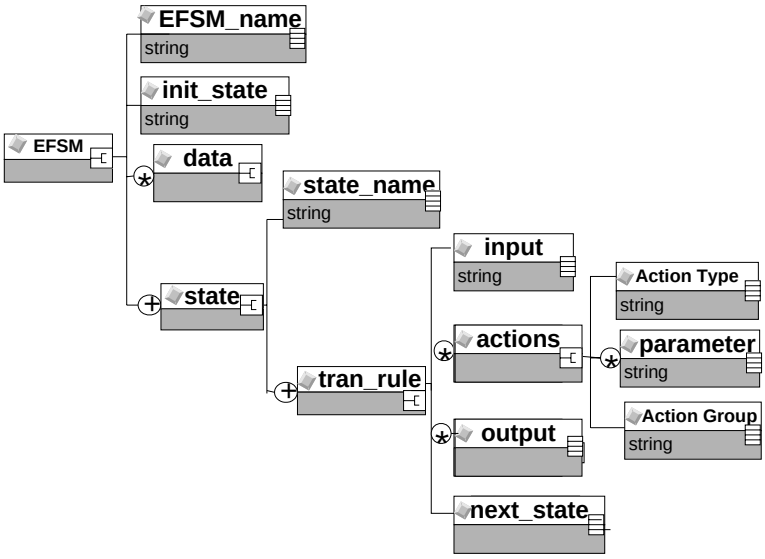


Figure 12 - EFSM-based XML manuscript data structure (cf. PAPER D and F).

The benefits of introducing the concepts of Action Library, Action Type, Action Group and Capability Category are mainly for achieving platform and implementation independence, as well as more flexibility and better reusability in service design. At the same time, they keep the service behaviour specifications short, clean and abstract from how it would eventually be implemented in different end-user devices.

C3: Preparation for Service Verification

This contribution addresses the problem statement **P1.3**. In an adaptable service system, new services and components are introduced into systems, where other services and components may already be running. Service verification is an important approach to ensure that components are introduced and assembled in a dynamic and error-free way. Service verification can be based on global verification or compositional verification. Global verification has limited applicability for complex systems such as adaptable service systems due to state explosions. To overcome this problem, compositional verification decomposes the service system and provides isolated verification of the decomposed parts.

Compositional verification based on an interface type language [CFN04] is adopted in the thesis. In order to utilize the compositional verification techniques provided, the service component behaviour needs to be extracted and translated to the interface type language. PAPER E presents the translation mechanism from EFSM-based XML manuscript to the interface type language. The translation is based on projection and consists of two steps. It first makes projections that preserve the binding between the role sessions (i.e. the component-component interface) related to each service component by using hidden actions. Then remove the hidden action so a sound verification can take place.

5.3 Topic T2: Service Discovery

Service discovery is a core functionality to locate desired services in a distributed environment. In a dynamic system, the amount and variety of services continue to increase and the service components and devices that offer services can come and go all the time. Service discovery thus becomes an important and difficult task. Efficient and automated mechanisms and frameworks for service discovery are required.

Service discovery is a process of finding the desired service(s) by matching service descriptions against service requests. *Semantic service discovery* is a service discovery process based on ontology concepts. Semantic service discovery is considered in this thesis. There are two major research aspects of service discovery, namely: 1) the *semantic representation and matching procedure* for semantic service discovery, and 2) the *network infrastructure* for a service discovery system.

As for aspect 1, the matching procedure depends on the semantic representation of service. For example, [SBBA05] proposed an approach for service discovery based on high level service goals comparison, where service goals are expressed in terms of service predicates. The service goal comparison is based on the matching of association class names, complementary roles and evaluation of additional constraints.

As for aspect 2, the research on network infrastructure concentrates on the mechanism for the organization of directories and the communication between them.

C4: Semantic Service Discovery Procedure

This contribution addresses the problem statement **P2.1**. In this thesis, the focus is on aspects of reasoning in the discovery process, i.e., semantic representation that machines can operate on as well as automatic matching based on semantic similarity of the service description elements.

PAPER A proposes a semantic service discovery procedure based on the integrated semantic service description defined in Figure 11, where all the elements in the description are used for semantic matching during service discovery. The procedure considers the matching of both functional and non-functional properties. It is carried out by a Reasoning Machine (RM), and consists of both ontological inference and rule-based reasoning. The ontological inference compares the semantic similarity of the concepts in the service descriptions and requests, while the rule-based reasoning is used to apply further constraints for the discovery procedure.

An RM-based prototype for semantic service discovery has been implemented. To exploit the reasoning power of the RM, suitable rules need to be designed. The lesson learnt from the design and implementation of the RM-based prototype is that the accuracy of the results from the semantic service discovery procedure depends on the good design of such rules, which needs accurate knowledge of the services and applications.

The discovery procedure is a generic one, applicable to both centralized and distributed environment. In the thesis (PAPER B and PAPER C), it is applied to an autonomous environment with distributed, self-organizing and scale-free communication (cf. contribution **C5**).

C5: Super-peer SON-based Service Discovery System

This contribution addresses the problem statement **P2.2**. A large scale service discovery system consisting of autonomous directories is considered. P2P based systems have been proposed as promising solutions providing scalability and autonomy [PSNS03][SMK01]. The challenges in such P2P systems are: (a) finding the right peers to query and (b) efficient routing of messages [HS04].

This thesis aims to apply appropriate P2P architectures as the network infrastructure for service discovery systems. It focuses on the solutions to an important question: *how to locate services efficiently in a large–scale system with a huge amount of a large variety of services*. With regard to the above mentioned challenge (a), the system needs to find relevant directories for a discovery request. For this, Semantic Overlay Networks (SONs) are used to logically connect directories with semantically similar contents. To discover a service, the service requests only need to be sent to relevant SONs. With regard to challenge (b), a super-peer structure is applied to the various SONs. The routing of service requests consists of the inter-SON communication and intra-SON communication assisted by super-peers.

Figure 13 illustrates the architecture for super-peer SON service discovery system. PAPER B and PAPER C describe the required functionality for the organization of directories into such super-peer SONs, and the use of such SONs for efficient service discovery as well as efficient SON management. The integrated semantic service discovery procedure proposed in PAPER A is adopted for semantic matching on selected directories. The service ontology defined in PAPER A is extended with one more concept: service category. The ontology-based semantics is used for two purposes: *organizing directories into SONs based on service categories* (a finer classification than domain), and *semantic annotation of service descriptions*. The semantics is added at two levels, i.e., at the level of the *directories* and at the level of *individual service* (i.e. service description) in each directory by using ontologies.

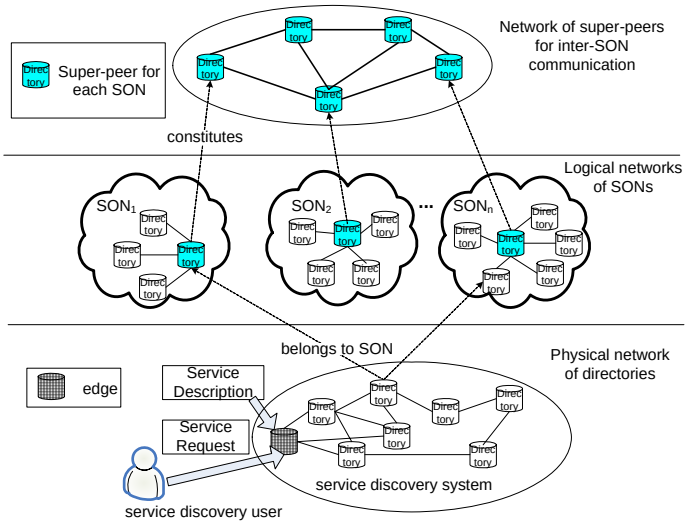


Figure 13 - Super-peer SON service discovery system architecture (cf. PAPER C).

A self-organizing process is applied for the construction and maintenance of such super-peer SONs based on an autonomous super-peer selection algorithm. The system

can dynamically determine and select how many super-peers are needed in the system. If one super-peer fails, new super-peer can be selected to reorganize the system. Such mechanism and the P2P-based architecture is an approach to realize the robustness and survivability property.

Evaluation based on simulations has been conducted for the super-peer based SON system. The super-peer SON system has been compared with a service discovery system based on a random overlay network. The results indicate efficient service discovery (in terms of recall, messages-per-request and hops-per-request) and efficient SON management (in terms of self-organization time, management procedure overhead and load factor).

The super-peer selection algorithm is described in APPENDIX A. The simulator and simulation model is described in APPENDIX B. Additional simulation results are provided in APPENDIX C.

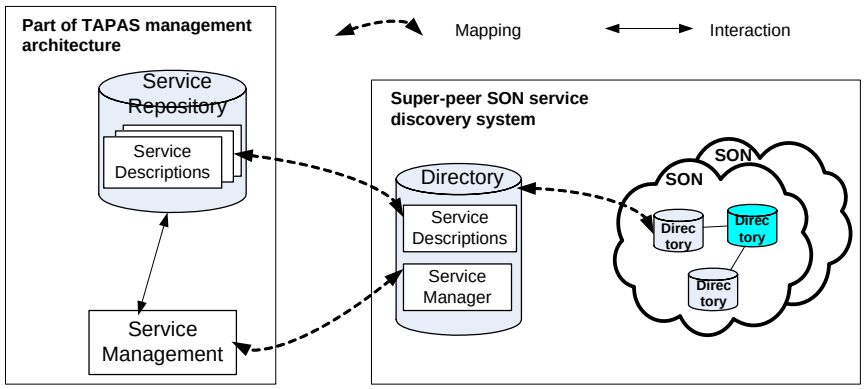


Figure 14 - Relation of the service discovery system and adaptable service systems.

Figure 14 illustrates the relations of the service discovery system and adaptable service systems (e.g. TAPAS-based systems). The TAPAS management architecture is a high-level conceptual architecture independent of implementation and network distribution. The proposed service discovery system architecture is a realization of service discovery functionality based on TAPAS management architecture with regard to network distribution. The service discovery system is targeted for service discovery both within adaptable service systems and between adaptable and non-adaptable service systems. A directory in the service discovery system can be a service repository from an adaptable service system, or from a non-adaptable service system. An adaptable service system can provide one or several directories for service discovery. A directory consists of a set of service descriptions and a service manager. The service descriptions in the adaptable service system will be mapped to the service descriptions contained in a directory in the service discovery system. The service manager carries out the service discovery-related functionality as well as SON management functionality. Concerning the discovery functionality, a service manager will (de-)register a service, carry out the local semantic matching, and collaborate in the dissemination of discovery messages in the discovery system (cf. PAPER B and C). A service is registered in a directory using *Register()* function defined in PAPER B and C, which means to store its service description in the directory. This thesis assumes that the changes in the services can be reflected in the service descriptions. Join policies can then be applied by the service

manager to adapt the SON membership dynamically according to runtime situation (cf. PAPER C).

5.4 Topic T3: Service Instantiation

Service instantiation provides the physical service to the service user. Since in adaptable service systems, the system resources allocated to service components are varied and may change all the time, the challenge is how to instantiate the service according to the available capabilities and status information (**P3**).

C6: Manuscript Execution Support – State Machine Interpreter

This contribution addresses the problem statement **P3**. This PhD work in this regard is based on TAPAS core platform (cf. Section 4.1). A general framework of service system has been proposed based on the physical service representation, i.e., the EFSM-based XML manuscript (PAPER D and PAPER F). Specifically, a tool (i.e. State Machine Interpreter) has been implemented for the instantiation of TAPAS-based services.

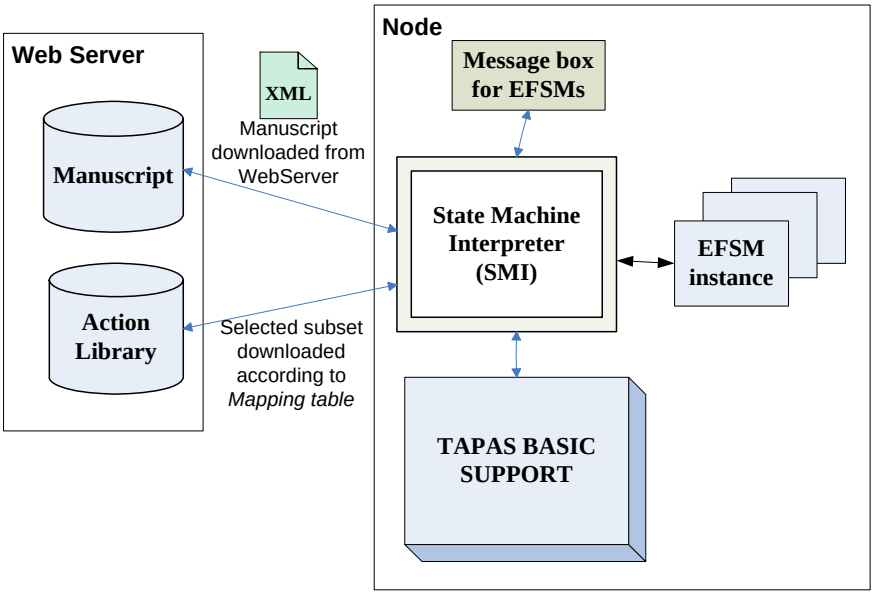


Figure 15 - Engineering model for service instantiation.

The engineering model for service instantiation is given in Figure 15. As stated in Section 5.2, the behaviour specification has been separated into Manuscripts and the Action Library, so that behaviour can be specified in platform independent and platform-dependent parts. Mapping tables are required to link the action definitions in the service behaviour specification to the executable codes stored in Action Library. The instantiation of the XML manuscript is based on dynamically selecting the execution codes, i.e., by downloading a subset of Action Library based on the Mapping table. The State Machine Interpreter (SMI) instantiates and executes the manuscript by comparing the offered Capability Categories in the Mapping table and the required Capability Categories in the Action Library, and invoking the action codes with

matching action types and capability requirements. In other words, the codes will be selected when the offered Capability Category matches the required Capability Category for the specific action. This mechanism scales as Action Library can be expanded without expanding the manuscript definition and only a subset of Action Library needs to be downloaded. The functionality and the implementation of the SMI are described in [JA03]. The generation of Mapping tables will be discussed in Section 5.5.

5.5 Topic T4: Service Adaptation

Adaptable service systems need *adaptability functionality* to handle various dynamic changes. Such adaptability functionality should be performed without stopping or interrupting other parts of the system which are not directly affected. The principles given in [KM90] are considered as a basis for any dynamic change management, which are also followed in the work reported in this thesis. The changes should be specified based on separation of concerns, i.e. separate the structural aspects (i.e. configuration) from the functional aspects (i.e. algorithms, protocols and states of the applications) [KM90].

Examples of the adaptability functionality include: changing a specific role played by an actor, moving a role figure from one node to another, modifying the execution of a manuscript definition for a role figure to adapt to changes in the environment without changing the definitions of the service system, reallocating and re-initializing capabilities within existing service systems. In TAPAS, the mechanisms for the various adaptability functionality are realized by different components in the TAPAS management architecture.

The adaptability functionality can be applied to realize conceptual service adaptation or physical service adaptation. This thesis addresses physical service adaptation.

C7: Physical Service Adaptation

This contribution addresses the problem statement **P4**. PAPER F and PAPER G propose an approach for physical service adaptation based on a RM-based mechanism for dynamic code selection. It dynamically changes manuscripts execution by selecting and instantiating physical service definitions (i.e. the parameterized EFSM-based XML manuscripts and corresponding action codes for different capability and status requirements) according to the runtime capability and status information. It enables different execution codes to be selected dynamically for the same behaviour type specification.

Adaptability implies *intelligence*, i.e. the ability to make decisions according to the dynamic changes in the system and environment. The adaptability is achieved by a RM. The RM has the intelligence for selection and adaptation because it has knowledge of what are the conditions and what are the actions to choose. Allowing RM to dynamically reason about the different implementation codes based on the availability of system resources introduces a great degree of flexibility into the manner how the service behaviour specification is carried out.

Given an adaptation request, a RM can select EFSM-based XML manuscripts and generate the Mapping tables according to the Selection Rules and Mapping Rules defined. In addition, parts of the manuscript specification can be changed (computed

dynamically) according to the available capability and status (PAPER G). The problem of how to generate different adaptation requests according to traffic situation and failure states is however not considered.

Figure 16 illustrates the generation of the Mapping table. Rules for mapping Action Groups to Capability Categories are defined in Mapping Rules. The rule-based RM computes the Mapping table based on the instantaneous Capability and Status information of the execution environment as well as the Mapping Rules. By assigning action types to a few Action Groups in the manuscript and action codes to Capability Categories in the Action Library, only a few Action Groups are needed to be mapped to Capability Categories, thereby, the amount of computation tasks for RM is reduced. At the same time, the resulting Mapping table is far shorter than the direct mapping from action types, thus greatly saves the communication overhead.

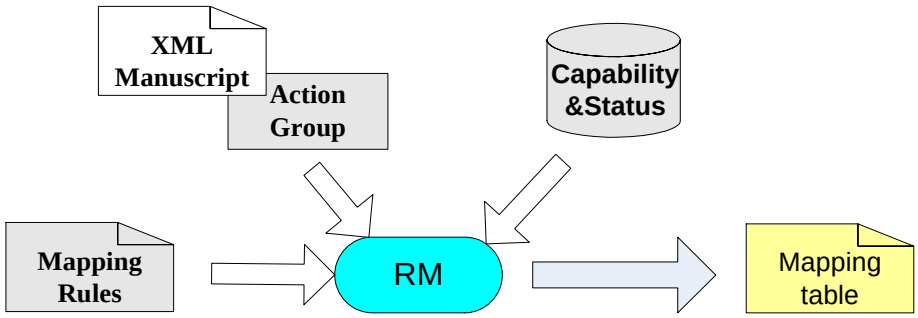


Figure 16 - Generation of the Mapping table.

C8: Dynamic Service Management Framework

The approach for physical service adaptation is realized by a framework for dynamic service management, which provides an overall solution by considering service specification, selection and adaptation. The service specification is based on the modifiable and parameterized behaviour patterns with generalized action types (the XML manuscript described in Section 5.2). Service selection refers to the selection of optimal code for execution, and is based on the selection of manuscripts and computation of Mapping table according to the available capabilities and status information. Service adaptation is achieved by dynamically selecting the execution codes in the manuscripts according to the changes in the capability and status information. The service selection and adaptation is based on the mechanism described in this Section for "C7: Physical service adaptation".

Figure 17 illustrates the dynamic service management framework. The Service Manager (SM) is a RM-based selection engine, which can select manuscripts and generate Mapping tables according to service requests and runtime capability and status information. The SMI is responsible for instantiating the manuscripts according to the manuscripts and Mapping tables sent by SM.

C9: Prototypes and Simulations

The proposed frameworks and mechanisms in this thesis have been evaluated and validated by prototypes and simulations. The prototypes have been implemented to

demonstrate the applicability of the proposed framework and mechanisms. In PAPER A, the integrated semantic service discovery procedure is carried out by an XET-based RM [Sup07][AWW02]. In PAPER D, a tool of SMI is implemented as an extension of the original TAPAS core platform. In PAPER E, the translation mechanism is demonstrated on a TAPAS-based example application – TeleSchool. In PAPER F and G, the dynamic service management framework is prototyped based on TAPAS core platform and XET-based RM, and example scenarios based on TeleSchool are described. In addition, simulations are used to give more detailed results with respect to system performance. In PAPER B and C, simulations have been carried out to evaluate the super-peer SON service discovery system with respect to the defined evaluation measures.

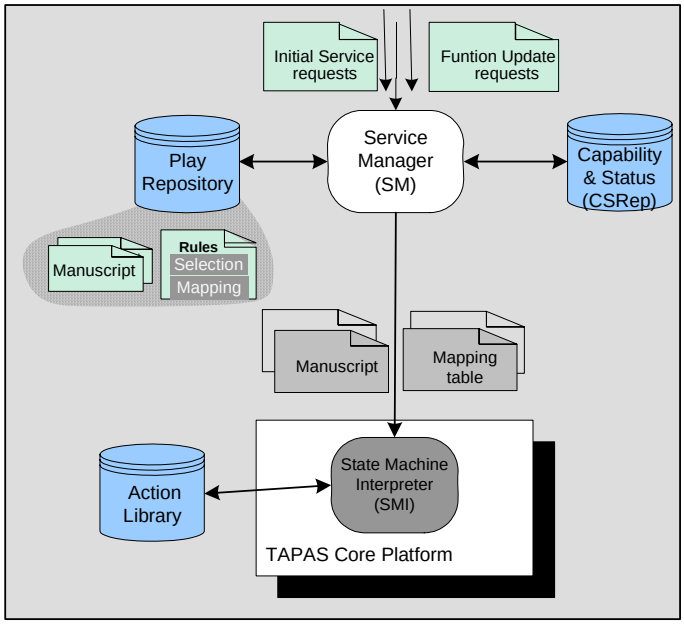


Figure 17 - Dynamic service management framework (cf. PAPER F).

5.6 The Realization of the Problem Statements

This section summarizes the relationship between research contributions and the problem statements.

Concerning **P1**, data models and representations for conceptual services and physical services have been proposed. For conceptual services (**P1.1**), a service ontology model (Figure 5) is defined and representation is based on Web Services and Semantic Web languages, such as WSDL, OWL and WS-Policy. Ontology is used as the basis to add semantics to service descriptions and requirements. This model and representation is defined in PAPER A, and used in PAPER A, B and C as a basis for service discovery. For physical services (**P1.2**), EFSM data model and XML representation (Figure 12) is applied. For **P1.3**, mechanism for translation from such XML service behaviour representation to interface behaviour representation based on a behaviour type language has been proposed to facilitate compositional verification. Such translation mechanism is described in PAPER E. The application of XML and ontology-based languages as well as the physical data model support the rearrangement flexibility (core property **A1)**.

Concerning **P2**, an integrated semantic service discovery procedure is proposed in PAPER A for the realization of **P2.1**, and a super-peer SON service discovery system (Figure 13) for efficient service discovery is described in PAPER B and C for the realization of **P2.2**. The *accurate* and *automatic* service discovery (**P2.1**) is enabled by 1) the matching of both the functional and non-functional properties for discovery, 2) the application of ontologies for formal and accurate semantic representation of service descriptions and requests, and 3) the application of RM for ontological inference and rule-based reasoning. Such discovery procedure supports the core property **A1**. On the other hand, to discover services *efficiently* in a large-scale service system (**P2.2**), a super-peer SON-based service discovery system is proposed. It applies the above discovery procedure to a large-scale, autonomous, self-organizing network infrastructure. Such P2P-based architecture and the super-peer selection algorithm for constructing and maintaining the super-peer SONs support the robustness and survivability property (core property **A2**).

Concerning **P3**, the physical service representation (XML manuscript) is instantiated and executed by State Machine Interpreter (SMI), proposed in PAPER D and extended in PAPER F. The XML manuscripts can be downloaded dynamically from a web server as needed and instantiated according to the Mapping table. The Mapping table provides the linking of the required capabilities and status of the action codes and the available capabilities and status information. Such mechanism supports the core property **A1**.

Concerning **P4**, the flexibility in introducing new service or modifications to existing services is realized by a dynamic service management framework described in PAPER F and G, which considers service specification, service selection and service adaptation in an overall framework. The service specification is based on the XML manuscript defined in PAPER D and extended in PAPER F. It separates the specification and implementation of actions by the parameterized behaviour patterns defined in XML manuscripts, and the implementing action codes contained in Action Library. The selection and adaptation of service is RM-based, where the RM can dynamically compute the Mapping table as well as select appropriate manuscripts according to the capability and status information. In addition, parts of the manuscript specification can be changed (computed dynamically) according to the available capabilities and status (PAPER G). The rearrangement flexibility is realized by the separation of the specification and implementation of actions, the RM-based computation of Mapping tables, and the dynamic instantiation based on such Mapping tables. Such framework thus supports the core property **A1**.

Concerning **P5**, the evaluation and validation by prototypes and simulations concludes that the above contributions meet the problem statements.

Figure 18 illustrates the relationship between papers, problem statements and contributions. As an example, **P1** has three sub-problems, **P1.1** – **P1.3**, each which is addressed by a contribution **C1** – **C3**, respectively and is dealt with in PAPER A, D and E, respectively. **P5** is related with **C9** and has been addressed in all papers.

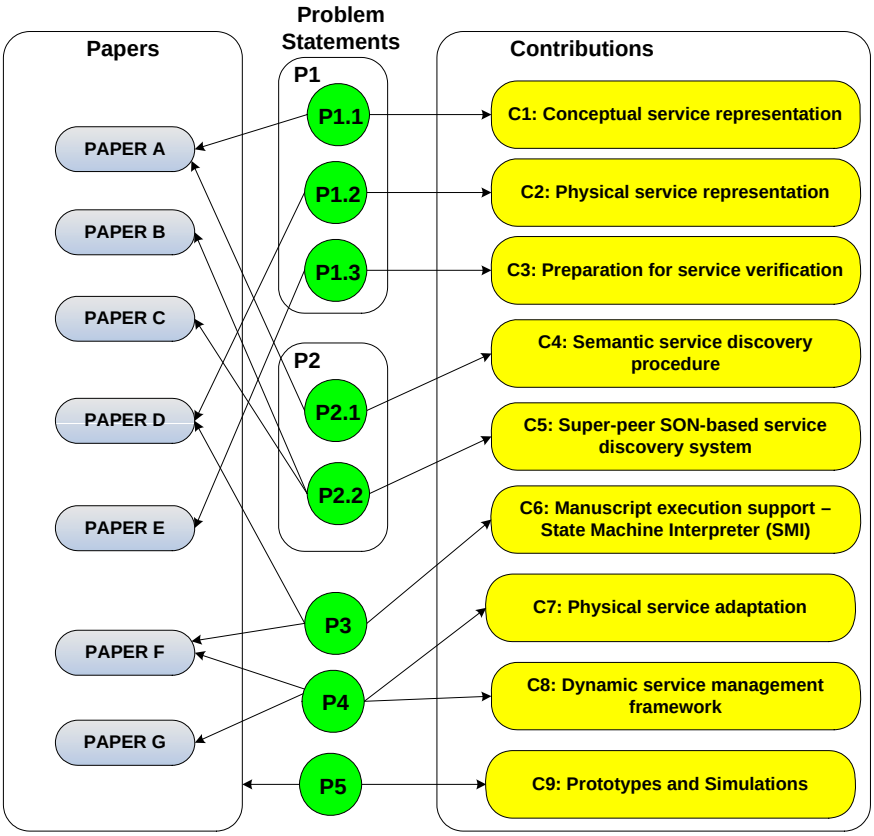


Figure 18 - Relationship between papers, problem statements and contributions.

5.7 Guidelines for Reading of Part II

Each paper included in Part II is self-contained and focuses on one or several topics. APPENDIX A, B and C provide additional descriptions about the algorithms, simulator and results for super-peer SON service discovery system, which help understand PAPER B and PAPER C and are an integral part for the service discovery topic. Since the topics covered by the papers and appendices are linked, it is recommended to follow the suggested reading order as illustrated in Figure 19.

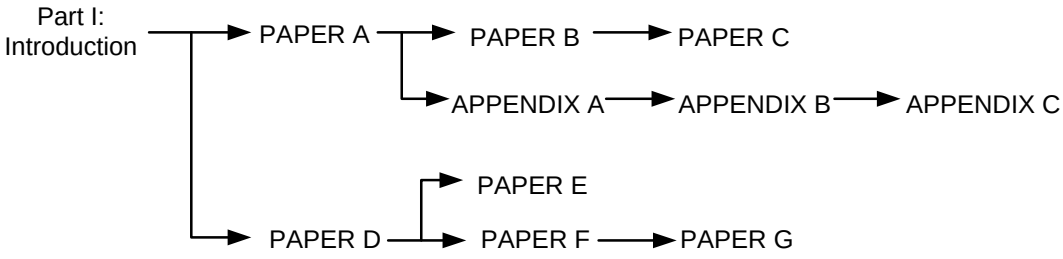


Figure 19 - Suggested paper reading order.

6. Research Methodology

The research presented in this thesis can be mainly characterized as research in the *constructive* context [Bri91]. The research process followed the *design science research methodology*, also called *constructive research methodology*. The design science aims at creation of new *artefacts*, i.e. things created by humans that can bring some utilities. The artefacts can be generally classified as *constructs (or concepts), models, methods and instantiations* [MS95]. This thesis deals with the following specific artefacts: concepts, models, algorithms, mechanisms and frameworks. A general methodology of design research is proposed in [VK04], which has been followed in the work presented in this thesis. The research cycle is depicted in Figure 20, and consists of the iterations of the following phases:

- *Problem formulation*. A set of problem statements has been identified in this phase. Such problem formulation is usually the result from literature review, which evaluates and analyzes the current state-of-art. At the same time, the research context is also clarified to understand the requirements of the target systems.
- *Suggestion and development.* This is a phase that results in new functionality based on a novel configuration of existing/new elements. Since several different research topics have been investigated, several different artefacts have been developed in this thesis. Examples of artefact include: concepts and models for service representation, methods (algorithm and procedure) for semantic matching and translation of XML manuscript to the behavioural type language, a framework for dynamic service management, a tool for service instantiation, a system model for super-peer SON service discovery system as well as the algorithm for super-peer SONs construction and maintenance.
- *Evaluation and validation*. Methods are used in this phase to evaluate and validate the results from the suggestion and development phase. Two methods are used in this thesis, namely: prototypes and simulations. The choice of the methods depends on the research focus. When the focus is on the demonstration of functionality, validation by prototypes is used. When the system performance is a main concern and real world execution is not practical, simulations are selected.
    - *Prototypes:* Most of the proposed artefacts are implemented as prototypes based on TAPAS (PAPER A, D, E, F and G). The main purpose is to demonstrate the implementation possibility and to validate the feature applicability by implementing the various mechanisms, algorithms and frameworks based on TAPAS.
    - *Simulations:* For large-scale systems, simulation is used to study the system behaviour. Such approach is used for evaluation of system performance of super-peer SON service discovery system in PAPER B and C. Evaluation measures are defined and simulation models are constructed with tuneable parameters. Simulations have been based either on simulation programs written by the author of this thesis or on

an open-source simulator – PeerSim [Pee07]. The simulation results are used as indications of system performance.

- *Result discussion.* Results from the evaluation and validation are discussed and conclusions are drawn accordingly. Sometimes the results and additional knowledge gained in the suggestion and development phase may be used as new inputs for another round of suggestion and development. In addition, new problem statements may be identified and lead to iteration of a new research sequence.

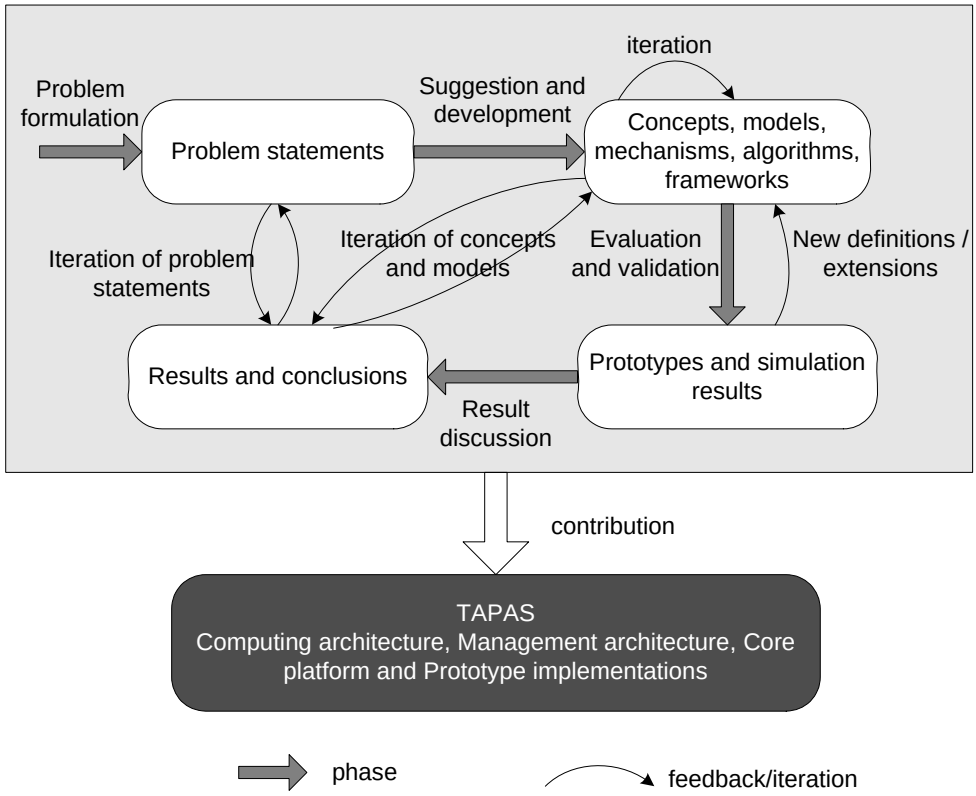


Figure 20 - Research cycle.

7. Summary of Papers

This section summarizes the seven papers included in Part II of this thesis and presents the main contributions of each of the papers (cf. section 5).

PAPER A: *An Approach to Integrated Semantic Service Discovery*

In a distributed service environment, service discovery is a core functionality to locate the desired services. This paper deals with the research problem of the semantic representation and matching procedure for service discovery. Service discovery has been a hot topic in the last years and many different approaches have been proposed. One limitation of many available approaches is that they either lack of the semantic representation of service behaviour in service descriptions, or make no use of them in service discovery process. Another limitation is that many approaches limit their discovery to functional-properties only, without considering non-functional properties during the process.

Ontologies are the basis for adding semantic expressiveness to service descriptions and requirements. Semantic service discovery is a service discovery process based on ontology concepts. This paper proposes an integrated semantic service discovery approach based on a service ontology. The service ontology defines concepts related to both the functional and non-functional properties of service. An integrated semantic service description is defined based on the service ontology. Such service descriptions are the conceptual service representation. Functional properties are described in terms of operations, inputs, outputs, preconditions and effects, while non-functional properties are specified as service parameters, QoS parameters and policies, which consist of business policies, QoS policies and context policies. Semantics are added to the service descriptions based on ontologies. Service requests are integrated with context information and personalized ranking criteria. An integrated semantic service discovery procedure based on the integrated semantic service description for semantic matching of both functional and non-functional properties is presented. Ontological inference and rule-based reasoning are applied for automatic and accurate discovery. A prototype discovery system (demonstrator) is implemented based on an XET-based reasoning machine to carry out the above discovery procedure.

Our work is based on widely accepted standards in Web Services and Semantic Web, i.e., WSDL, OWL and WS-Policy. The integrated semantic discovery approach proposed is a generic one, and can be applied to a centralized or distributed environment.

The main research topics of the paper are *conceptual service representation* and *service discovery (semantic representation and matching procedure)*. The problem statements addressed are **P1.1** and **P2.1**. The major contributions of the paper are:

- ***C1*** *{ Conceptual service representation}*
- ***C4*** *{Semantic service discovery procedure}*
- ***C9*** *{Prototypes}.*

PAPER B: *A Self-organizing Service Discovery System Based on Semantic Overlay Networks*

This paper has focus on the network infrastructure for service discovery system. It provides an approach to applying the semantic matching procedure proposed in PAPER A on a large-scale network infrastructure.

Currently, several service discovery protocols are available for enterprise and home network. These approaches rely on centralized entities for discovery or search entire local network by broadcast or multicast, thus are not suitable for a large-scale system with a huge amount of a large variety of services. An important research objective is thus the location of services in such a large-scale system *efficiently, i.e. to answer service requests fast with low overhead.*

A network of autonomous directories forming a large-scale distributed service discovery system is considered. We propose to apply semantic service discovery and to organize directories into Semantic Overlay Networks (SON) based on service ontology. The various SONs are further organized into super-peer networking structures. Service discovery based on SONs is an iterative process that can consist of several loops. Each loop has two steps. In Step 1, SONs which may contain relevant directories are selected. In Step 2, all the directories in these SONs are searched based on semantic matching. The result of PAPER A, i.e., the integrated semantic service discovery approach that can carry out semantic matching, has been proposed for Step 2. In each loop, the results are checked to see if enough number of matches is found. If not, more SONs are selected in new loops and until enough number of matches is obtained.

A generic model for SON-based service discovery system is presented, and a super-peer SON service discovery system is described. The focus is on the functionality for the organization of directories into SONs and the self-organizing construction and maintenance of super-peer SON networks. Different policies are applied as constraints on the system functionality. Aggregation and assignment policies are applied to ensure the size and number of SONs can be small. The number of SONs and SON membership can change dynamically by applying policies on each directory. Iterative selection of SONs for discovery enables high probability searching. The proposed system has been evaluated by simulations. The results indicate that such super-peer based SON system can improve discovery efficiency. The proposed functionality gives a small number of SONs with small SON size and requires a small management procedure overhead. The results also indicate short self-organization time both in initial SON construction situations and in situations when nodes are leaving. Moreover, discovery procedure overhead is significantly reduced compared to a system based on a random overlay network (i.e. pure, unstructured P2P system).

The main topic of the paper is *service discovery (network infrastructure)*. The problem statement addressed is **P2.2**. Contributions of the paper are:

- ***C5*** *{Super-peer SON service discovery system}*
- ***C9*** *{Simulations}.*

PAPER C: *Efficient Service Discovery System Based on Semantic Overlay Networks*

This paper extends the work of PAPER B. An efficient service discovery system is characterized by efficient service discovery and efficient SON management. Efficient service discovery means high recall, small number of messages-per-request (i.e. discovery procedure overhead) and small number of hops-per-request. Efficient SON management is characterized by small management procedure overhead, small load factor and short self-organization time. The effect of service distribution and join policies on system performance has been studied and more simulations have been carried out for the service discovery procedure. Specifically, two cases for service distribution have been simulated, namely: 1) when services are clustered and 2) when services are evenly distributed. Simulations indicate that such super-peer based SON system can significantly improve discovery efficiency by providing high recall with small hops-per-request value and significantly reduced messages-per-request value. Moreover, such system requires only a small management procedure overhead and the self-organization time is short both for SONs initial construction and reconstruction under dynamic node joining and leaving situations. Simulations also indicate a small average load factor. The results, however, indicate that a SON system has better performance when services are clustered than when services are evenly distributed.

APPENDIX A, B and C provide additional information for understanding PAPER B and PAPER C, and are integral part of service discovery topic. APPENDIX A describes the super-peer selection algorithms for super-peer SONs. APPENDIX B provides information on the PeerSim simulator. APPENDIX C gives additional simulation results.

The main topic of the paper is *service discovery (network infrastructure)*. The problem statement addressed is **P2.2**. Contributions of the paper are:

- ***C5*** *{Super-peer SON-based service discovery system}*
- ***C9*** *{Simulations}.*

PAPER D: *XML-based Dynamic Service Behaviour Representation*

Modelling of behaviour is important for dynamic adaptable service development and deployment. An implementation language independent framework for behaviour description in XML is presented in this paper. The service behaviour is an EFSM-based functionality defined in the manuscript. The basic XML data structure for such EFSM-based behaviour is defined. The framework is based on a State Machine Interpreter (SMI)[3] (XML engine), which is the interface between XML-represented behaviour description and runtime environment, and can interpret and execute XML-represented EFSM behaviour. Functionalities for a SMI are provided in the paper.

The challenge of using XML for service execution representation is how to represent the dynamic service logic, as XML basically is a structure representation language. Our dynamic behaviour representation is made possible by utilizing the inherent characteristics of the TAPAS computing architecture. Capability is considered in

[3] In the paper, it is called FSM interpreter. In subsequent work and papers, State Machine Interpreter or SMI is used. To be consistent, the name of SMI has been used throughout the first part.

service modelling as part of the service specification, i.e. service behaviour is defined by plug-and-play manuscripts and node inherent capabilities.

This model has been implemented and integrated into TAPAS platform. An SMI for TAPAS platform has been implemented as an example of realizing the XML-based service system model. One novel design is the separation of behaviour specification into *manuscripts* and *action library*, so that behaviour can be specified in platform-independent (generic) and platform-dependent parts (which can be extended or adapted according to the available capabilities and status information). In this way, the specification and implementation of actions is separated.

In addition, using XML for service execution representation gives one integrated representation of all dynamic service related functionality in TAPAS.

The main topics of the paper are *physical service representation* and *service instantiation*. The problem statements addressed are **P1.2** and **P3**. Major contributions of the paper are:

- ***C2*** *{Physical service representation}.*
- ***C6*** *{Manuscript execution support – the SMI}*
- ***C9*** *{Prototypes}.*

PAPER E: *Automatic Translation of Service Specification to a Behavioural Type Language for Dynamic Service Verification*

Networked services are constituted by the structural and behaviour arrangement of service components. A service component is executed as an actor, which can download and execute different EFSM-based functionality. The functionality of an actor is denoted as its role, while a *role session* is a projection of the role with respect to the interaction with one other actor. In an adaptable service system, new services and components are introduced while other services and components may already be running. Service verification is therefore an important approach to ensure that components are introduced and assembled in a dynamic and error-free way.

There are basically two different approaches to service verification. One is global verification by modelling the composite behaviour of the whole service, which has limited applicability for complex systems due to state explosions. Another approach is the compositional verification by decomposition of the service system and isolated verification of the decomposed parts. Service components are naturally decomposed parts for such compositional verification. We propose an approach for verification of the services, based on interface verification techniques for the verification of the role sessions. The service component specifications used for actor execution are based on XML-based EFSM representations (i.e. the EFSM-based XML manuscripts defined in Paper D), while the verification of the role sessions (i.e. component-component interface) is based on a behaviour type language [CFN04]. This language has a sound theoretical basis, and provides formal framework for compositional verification of component based systems. Especially, it is used to avoid “message-not-understood” errors while plugging a new component. Rules are given for automatic translation from XML-based EFSM service specification to the behaviour type language applied. This translation first makes projection to the role session, using hidden actions. Those hidden

actions are then removed so a sound verification can take place. The automatic translation provides an efficient and reliable way to extract interface types. An experiment has been carried out based on an example application.

The main topic of the paper is *service representation (preparation for service verification)*. The problem statement addressed is **P1.3**. The major contributions of the paper are

- ***C3*** *{Preparation for service verification}*
- ***C9*** *{Prototypes}.*

PAPER F: *An Approach for Dynamic Service Management*

Managing dynamic changes in a service system requires the handling of the modifications and extensions of the system without stopping or disturbing its functionality. The solution must be based on a flexible service specification which can be adapted by a supporting runtime mechanism. For this purpose, a framework for dynamic service management with respect to service specification, selection and adaptation is proposed.

Service specification is based on modifiable and parameterized behaviour patterns with generalized action types (an extended EFSM-based XML manuscript of PAPER D). Service functionality is classified into *Action Groups* and *Capability Categories* according to the nature of actions and the dependability on capability respectively. Such service specifications can be instantiated by dynamically selecting the execution codes, or the subset of Action Library that include routines specific to the execution environment (i.e. based on Capability Categories), according to the execution environment context and the service requests. A mapping (i.e. Mapping table) is required to link the action definitions in the service specification to the executable codes stored in Action Library. Service adaptation is achieved by allowing these service specifications to be modified according to the dynamic changes in the executing environment, a process based on dynamic code selection realized by the dynamic mapping from Action Group to Capability Category according to the runtime context and capability information.

The dynamic service management framework extends the TAPAS core platform by providing the intelligence and flexibility of dynamic service selection and adaptation. *Service Manager(SM)* is responsible for service selection and adaptation processes and makes decisions based on a *rule-based reasoning machine*. Rules for selecting manuscripts and mapping Action Groups to the corresponding Capability Categories are defined as *Selection Rules* and *Mapping Rules* respectively. The service requests (containing also context information), the instantaneous Capability and Status information of the execution environment, as well as the rules are inputs for the reasoning machine to compute the Mapping table and select the specific manuscript for execution. A prototype reasoning and selection engine for Service Manager based on XDD [WAAN01] and XET is implemented to illustrate the feasibility of the framework proposed. An *SMI* manages the execution of the manuscripts and the linking of action definitions with their implementation in the Action Library based on the Mapping table. An example that exploits these features is also presented.

The main topic of the paper is *physical service adaptation*. The problem statement addressed is **P4**. Major contributions of the paper include:

- ***C7*** *{Physical service adaptation},*
- ***C8*** *{Dynamic service management framework},*
- ***C9*** *{Prototypes}.*

In addition, the paper has the following minor contributions:

- ***C2*** *{Physical service representation}: extension to the EFSM-based XML manuscript data model defined in PAPER D.*
- ***C6*** *{Manuscript execution support – SMI}**:** extension of SMI to support the execution of the extended XML manuscripts.*

PAPER G*: An XML-based Framework for Dynamic Service management*

This paper extends the work presented on PAPER F. Role Figures are the constituents of the architecture that is used to provide a basis for service specification and instantiation. Service specification is the Role Figure behaviour specifications, which has three forms, namely (static) Role-Figure Specification, instantiated Role-Figure specification and calculated Role-Figure Specification. An extension of the EFSM-based manuscript data structure, i.e. *substate* structure, has been proposed to allow for more flexibility and adaptability in specification. The XML-based framework for dynamic service management handles the selection and computation of these specifications. Web services technology is used to manage the availability and communication of service components. To allow for a decentralized computation, SMI can calculate the Role-Figure Specification in the local node where it will be executed, instead of the SM. This feature can solve problems such as those related to over-loaded SM or congested network. An example scenario of Role-Figure move request is illustrated. This work extends TAPAS core platform with Web Services communication routines and node registry capabilities.

Both PAPER F and PAPER G consider physical service adaptation, where manuscripts are dynamically selected and instantiated according to the given service adaptation requests. The dynamic generation of such adaptation requests according to traffic situation and failure states is not considered.

The main topic of the paper is *physical service adaptation*. The problem statement addressed is **P4**. The major contributions of the paper are:

- ***C7*** *{Physical service adaptation}.*
- ***C8*** *{Dynamic service management framework}.*
- ***C9*** *{Prototypes}.*

In addition, this paper has the following minor contribution:

- ***C2*** *{Physical service representation}: extension to EFSM-based XML manuscript data structure.*

8. Summary, conclusions and future work

8.1 Summary of Results

The context for this thesis is service engineering and service management in adaptable service systems. The main research objectives are to *specify, construct, evaluate and validate applicable concepts, models, mechanisms, algorithms and frameworks for service engineering and service management in adaptable service systems.* Five problem statements P1-P5 (including sub-problems) have been defined in Section 3. These problem statements are related to the following four research topics: *T1: Service representation, T2: Service discovery, T3: Service instantiation* and *T4: Service adaptation*.

The papers in Part II address the research topics and problem statements with proposed concepts, models, algorithms, mechanisms and frameworks. They constitute the following contributions:

- **C1:** Conceptual service representation
- **C2:** Physical service representation
- **C3:** Preparation for service verification
- **C4:** Semantic service discovery procedure
- **C5:** Super-peer Semantic Overlay Network (SON)-based service discovery system
- **C6:** Manuscript execution support – State Machine Interpreter
- **C7:** Physical service adaptation
- **C8:** Dynamic service management framework
- **C9:** Prototypes and simulations

For topic T1: Service representation, the contributions are C1-C3. This thesis focuses on the representation of conceptual services and physical services. An integrated semantic service description is proposed for high-level conceptual service representation and is represented using Web Services and Semantic Web languages. The service ontology defines a model for procedure-based functionality and non-functional properties. Such semantic-annotated service description is the basis for semantic matching procedure in service discovery. On the other hand, XML manuscript is the physical service representation. An EFSM-based XML manuscript data model is defined based on modifiable and parameterized behaviour patterns. Such manuscript data model is the basis for service instantiation and adaptation. In addition, in order to utilize compositional verification techniques based on an interface type language, automatic translation from the EFSM-based XML manuscript to the interface type language has been provided based on projection. Projection technique is applied during the translation process.

For topic T2: Service discovery, the contributions are C4-C5. Two aspects of service discovery are considered, namely: 1) the semantic representation and matching procedure for semantic service discovery, and 2) the network infrastructure for a large-scale service discovery system. For aspect 1, an integrated semantic discovery

procedure based on semantic-annotated service descriptions is proposed for semantic matching of both functional and non-functional properties. Such procedure consists of ontological inference and rule-based reasoning and is carried out by a Reasoning Machine (RM). For aspect 2, a super-peer based SON service discovery system is proposed and functionality for efficient service discovery and efficient SON management is defined. The integrated semantic service discovery procedure proposed for aspect 1 is applied for semantic matching on selected directories (i.e. selected SONs). A self-organizing process based on an autonomous super-peer selection algorithm is applied for super-peer SONs construction and maintenance. The system performance is evaluated by simulations and the results indicate efficient service discovery and efficient SON management.

For topic T3: Service instantiation, the main contribution is C6. An execution support, namely the State Machine Interpreters (SMI), has been implemented to interpret and execute EFSM-based XML manuscripts.

For topic T4: Service adaptation, the contributions are C7 and C8. The physical service adaptation is realized by dynamic code selection, which can change the execution of service by dynamically select and instantiate EFSM-based XML manuscript according to runtime capability and status information. A RM-based dynamic service management framework integrating service specification (i.e. EFSM-based XML manuscript), selection (instantiation) and adaptation is proposed and prototyped. Selection and Mapping Rules are proposed and modelled.

For topics T1-T4, contribution C9 is used to evaluate and validate the proposed frameworks and mechanisms.

8.2 Conclusions

The work presented in this thesis consists of nine contributions C1-C9 that aim to meet the problem statements P1-P5. To conclude, these problem statements have been realized as follows:

For **P1**:

- P1.1 has been realized by C1. Conceptual services can be modelled based on a service ontology and represented using XML-based languages.
- P1.2 has been realized by C2. Physical services can be modelled as EFSMs and represented using XML. Separation of the physical service representation into parameterized behaviour patterns and Action Library with actual codes enables the flexibility of dynamic service adaptation according to available capabilities and status information.
- P1.3 has been realized by C3. The component interface behaviour can be extracted from the physical service representation using the translation mechanism proposed in PAPER E in order to apply compositional service verification.

For **P2**:

- P2.1 has been realized by C4. Automatic and accurate service discovery can be achieved by an integrated semantic service discovery procedure

considering both functional and non-functional properties and consisting of ontological inference and rule-based reasoning.

- P2.2 has been realized by C5. Super-peer SON-based service discovery system can be used to locate services efficiently in a large-scale service system.

For **P3**:

- P3 has been realized by C6. Services can be instantiated dynamically and according to available capabilities and status information using a SMI that can select actual execution codes based on Mapping tables.

For **P4**:

- P4 has been realized by C7 and C8. New service specifications or modifications to existing services can dynamically be introduced without interrupting the executing service based on the dynamic service management framework proposed in PAPER F and G.

For **P5**:

- P5 has been realized by C9. The evaluation and validation of the proposed frameworks and mechanisms by prototypes and simulations concludes that the contributions meet the above problem statements.

The solutions proposed in this thesis have contributed to TAPAS computing and management architecture and have been included in TAPAS.

Referring to core adaptability properties, the research presented in this thesis focuses primarily on the core property **A1**, i.e., rearrangement flexibility. This property is achieved by mechanisms that enable *interoperability* and *intelligence*. Interoperability is based on XML representation. In particular, semantic interoperability is realized by XML-based ontologies. Intelligence means the ability to make decisions according to the dynamic changes in the system and environment. In the thesis, a RM-based mechanism is the basis for achieving intelligence in service discovery and service adaptation. To exploit the reasoning power of the RM, suitable rules need to be designed. The lesson learnt from the design and implementation of the RM-based prototypes is that the accuracy of the results from the semantic service discovery procedure depends on the good design of such rules, which needs accurate knowledge of the services and applications.

The P2P-based architecture for service discovery system and the super-peer selection algorithm for constructing and maintaining the super-peer SONs support the robustness and survivability property (core property **A2**).

8.3 Directions of Future Work

This section presents proposals for further work.

Integrated framework for service management

This thesis has investigated four related service topics. A dynamic service management framework addressing service representation, service selection (including service instantiation) and service adaptation has been studied (PAPER F and G). However, service discovery has been investigated separately. Efforts are needed to integrate the respective mechanisms and procedures proposed for each of the four issues into one integrated framework. The performance of the integrated framework needs to be evaluated.

Generation of adaptation requests

The physical service adaptation solution proposed in this thesis assumes that changes in the system and environment can be detected and reflected as service adaptation requests sent to the service manager for appropriate handling. The generation of such requests according to the traffic situations and failure states is not considered in this thesis. It is an important yet complex functionality for service adaptation. This functionality needs not only the monitoring of the current situation (i.e. available capabilities and status information), but also detailed knowledge of the application. It is important to find a generic solution so that human intervention can be reduced in this process. Future work needs to analyze the aspects needed for the autonomous generation of such requests and integrate them into the physical service adaptation and dynamic management framework proposed by this PhD work.

Utilization of EFSM-based behaviour model in service discovery

The semantic representation and matching procedure proposed in the thesis work is based on a high-level conceptual service representation where service functionality is modelled as procedures. Sometimes it is not enough to find a service that offers some functionality. Instead, a service with specific behaviour is desired, which means the EFSM-based behaviour specification needs to be compared in service discovery. The EFSM-based behaviour model (i.e. a sequence of inputs and outputs) should be included in the conceptual service representation and used for matching in the integrated semantic service discovery procedure.

Conceptual service adaptation

As an important approach for conceptual service adaptation, mechanisms are needed for dynamic composition of services based on the conceptual service behaviour specifications. Such dynamic composition needs to be verified.

Platform dynamics

In the TAPAS systems, there are some specialized actors, such as Directors and service managers. These actors used to be allocated using predefined configuration rules. The self-organizing super-peer selection algorithm proposed in PAPER B and C can be applied to realize platform dynamics. The Directors and service managers can be considered as super-peers (i.e. nodes provide services to others) with special capabilities. This means that the system can dynamically determine and select how

many Directors or service managers are needed to manage the actors in the system. If one Director or service manager fails, new Director or service manager can be selected to reorganize the system. Such mechanism for platform dynamics can be considered as an approach to realize the robustness and survivability property.

Super-peer SON-based network infrastructure

The super-peer SON-based network infrastructure needs to be explored further. It is possible to extend the super-peer selection algorithm with mechanisms to introduce super-peer redundancy, i.e., with nodes being clients of multiple super-peers. Communications between the super-peers can be improved by mechanisms such as DHT-based structure. Extensive simulations are needed to evaluate its performance.

PART II: INCLUDED PAPERS

PAPER A: An Approach to Integrated Semantic Service Discovery

Shanshan Jiang and Finn Arve Aagesen

Published in
Proceedings of Autonomic Networking (AN'06)

Paris, France, September 27-29, 2006.

Lecture Notes in Computer Science (LNCS) 4195, pp. 159-171, 2006.

An Approach to Integrated Semantic Service Discovery

Shanshan Jiang

Finn Arve Aagesen

Abstract In a distributed service environment, service discovery is a core functionality to locate the desired services. We propose an integrated semantic service discovery approach based on ontology, which provides matching of functional and non-functional properties. Functional properties are described in terms of operations, inputs, outputs, preconditions and effects, while non-functional properties are specified as business policies, QoS properties and context policies. Ontological inference and rule-based reasoning are applied for automatic and accurate discovery.

1. Introduction

In a distributed service environment, service discovery is a core functionality to locate the desired services. *Service discovery* is a process of finding the desired service(s) by matching *service descriptions* against *service requests*. A *service description* provides service-related information which can be advertised by a service provider and searched during service discovery process. Such information usually includes functional properties and non-functional properties. In this paper, functional properties representing functionality of a service are modelled in terms of operations, inputs, outputs, preconditions and effects, while non-functional properties comprise business policies, Quality of Service (QoS) properties as well as context policies. QoS properties include QoS parameters and QoS policies. A *service request* represents user's service requirements, comprising requirements on functional and non-functional properties.

Ontologies are the basis for adding semantic expressiveness to service descriptions and requirements. An *ontology* is an explicit and formal specification of a shared conceptualization [19]. A service ontology is accordingly an explicit and formal specification of core concepts of the functional and non-functional properties of service. "A *domain ontology* (or *domain-specific ontology*) models a specific domain and represents the particular meanings of terms as they apply to that domain. An *upper ontology* is a model of the common objects that are generally applicable across a wide range of domain ontologies."[1] Ontological relations such as "is-subclass-of" or "part-of" are used for ontological inference.

Semantic service discovery is a service discovery process based on ontology concepts. By using ontology concepts defined in a service ontology expressively in a service description, semantics of the service description can be defined. These service descriptions are therefore expressive semantic descriptions. At the same time, by having both ontology-based descriptions and requirements, an ontology-enhanced reasoning engine (i.e. capable of ontological inference) can be used to locate services automatically and accurately. *Integrated semantic service discovery* is a semantic

[1] Wikipedia: http://en.wikipedia.org/wiki/Ontology_(computer_science).

service discovery process based on both functional and non-functional properties of the services.

Service discovery has been a hot topic in the last years, and many different approaches have been proposed. In Web Services technology, Web Services are described in WSDL (Web Services Description Language) [22] and advertised in UDDI (Universal Description, Discovery and Integration) [21] registries. UDDI provides only keyword-based discovery (e.g. service category or provider name) and makes no use of semantic information of service behaviour (e.g. semantics of operations, inputs and outputs) defined in the service descriptions during discovery. A number of protocols for service discovery have also been proposed, most notably, SLP (Service Location Protocol) [8], Jini [11], UPnP [7] and Salutation [4]. Service descriptions in these protocols are usually based on categories of predefined service types, interface types, attributed IDs and values, without expressive semantic descriptions to enable reasoning. Thus service discovery is restricted to simple keyword-based category and attribute matching. Other approaches based on Semantic Web technology, such as [13], provide semantic service discovery, but limit their discovery to functional properties only, without considering non-functional properties during the process. Integrated semantic service discovery, however, is important to achieve accurate and satisfactory discovery results.

In this paper, we propose an integrated semantic service discovery approach based on semantic-annotated WSDL [16]. Ontologies are defined in the Web Ontology Language, OWL [12]. Behavioural semantics are added to WSDL file by associating service functionality related elements with links to OWL-based service ontology. Non-functional properties are specified as QoS parameters and rule-based policies comprising business policies, QoS policies and context policies. Furthermore, WS-Policy Framework (Web Services Policy Framework) [6] and WS-PolicyAttachment (Web Services Policy Attachment) [5] are utilized to attach QoS parameters and policies to WSDL-based service descriptions. Service requirements are also expressed using ontology concepts. Based on them, an integrated semantic service discovery procedure is presented, which takes into account selection criteria based on business policies, QoS policies and context policies, as well as user defined service selection criteria in terms of overall QoS scores based on QoS parameters.

The rest of the paper is organized as follows: Sect. 2 discusses service description elements with focus on rule-based policy specifications. Based on it, an integrated semantic service discovery framework is presented in Sect. 3. Related work is discussed in Sect. 4, followed by summary and conclusions in Sect. 5.

2. Service Description Elements

A service description comprises the following elements:

- *Functionality description* in terms of operations, inputs, outputs, preconditions and effects.
- *QoS parameters* offered by the service.
- *Service parameters* such as location and other service specific parameters.
- *Policies* for service discovery, comprising business policies, QoS policies and context policies.

A policy is a rule applied in the decision making process. It is usually conditional criteria against which factual variables are evaluated to determine an appropriate action. Therefore, a rule-based policy representation is a natural choice for policy specification. Ontology-based policy description allows service providers and requestors to describe their policies with respect to a common ontology in terms of meaningful concepts and relations. Furthermore, benefit from semantic enrichment and ontological inference can be achieved. For service discovery process, policies will guide the optimal selection of desired services among several functionally equivalent services.

Formally, an ontology-based policy rule P can be defined as a tuple <r, o, s, c, a>, where r is a reference to a policy ontology, o denotes the organization P belongs to, s denotes the service P applied to, c the conditions, a the actions. WS-Policy framework [6] provides a general purpose model to describe and communicate policies of a Web Service. It places no restrictions on the language used to represent policy expressions. We use an ontology language, OWL, to express the policy based on the upper ontology for policy inspired by [18] and depicted in Fig. 1. The upper ontology defines the concepts used for policy specification and their relations. A *policy* belongs to an *organization* and is applied to a *service*. A *policy* has a *policy domain* and refers to a domain-specific *policy ontology*. A *policy* may have multiple *rule sets*, each of them defines a set of *rules*. The *rule sets* have *rule operators*, such as "ExactlyOne" to specify that only one rule set is applied at a time. Each *rule* is a conditions-and-actions statement, which specifies the *actions* to be performed when *conditions* are evaluated to TRUE. *Conditions* are specified in *expressions* while *actions* are associated with *operations* of a *service*. *Actions* may also have *expressions* and may require *conditions*. *Expressions* may have *attributes*, *literal values*, *operators* as well as *logical operators*. *Attributes* can be *service parameters*, *inputs*, *outputs* or *QoS parameters* of a service. Therefore, this upper ontology of policy has relations with service ontology, i.e. concepts in the gray area in Fig. 1 also belong to service ontology.

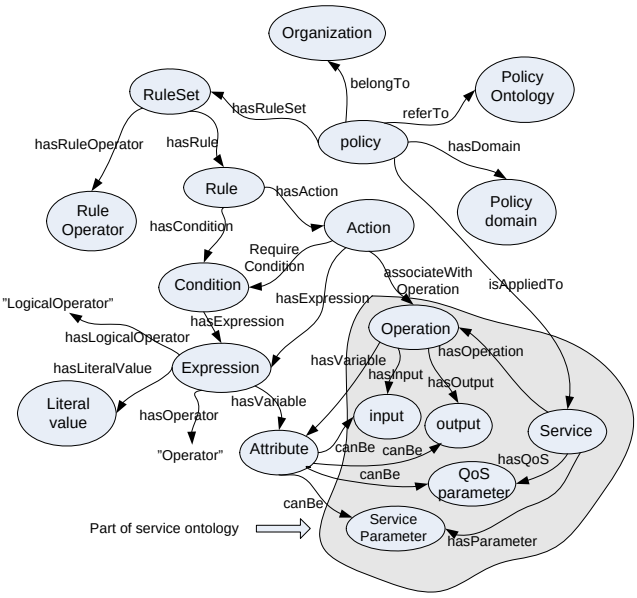


Fig. 1. Upper ontology for rule-based policy

2.1 Business Policies

Business policies are rules related to business concepts. They can be published associated with a service to constrain service discovery process. As an example, consider a delivery policy for an online bookstore, say *BookStoreA*, which specifies that if the number of copies ordered for a book is less than 50, then the delivery time will be within 5 days; if between 50 and 200, the order will be delivered within 10 days; otherwise, the inventory should be checked before an action is taken. The policy can be expressed in three rules as shown in Fig. 2.

Rule 1 **IF** (*numberOfCopies* ≤ 50)
THEN DeliverBooks (*deliveryDays* ≤ 5)
Rule 2 **IF** (50 < *numberOfCopies* ≤ 200)
THEN DeliverBooks (*deliveryDays* ≤ 10)
Rule 3 **IF** (*numberOfCopies* > 200)
THEN CheckInventoryFirst

Fig. 2. Example delivery policy for an online bookstore

A user who orders 100 copies of a textbook to be delivered within 15 days from online bookstores may select the *PurchaseBook* service from *BookStoreA* since his/her request can be matched by the second rule. Figure 3 shows part of the policy specification called *DeliveryPolicyBookStoreA* for an online bookstore *BookStoreA* based on the upper ontology defined in Fig. 1, which corresponds to **Rule 1** in Fig. 2. Note that the namespace *po:* refers to the upper ontology, while the namespace *sp:* refers to the domain-specific policy ontology. Attribute *numberOfCopies* is a service input, attribute *deliveryDays* is a service output, while operation *DeliverBookOperation* is a service operation.

```
xmlns:po="http://examplepolicy.com/policy.owl#"
xmlns:sp="http://ecommerce.com/policy.owl#"
xmlns="http://BookStoreA.com/PolicyRule.owl#"

<sp:DeliveryPolicy rdf:ID="DeliveryPolicyBookStoreA">
  <po:hasRuleSet rdf:ID="RuleSet1">
    <po:hasRuleOperator rdf:resource="po:ExactlyOne" />
    <po:hasRule>
      <po:Rule rdf:ID="Rule1">
        <po:hasCondition  rdf:resource="#CheckQuantity1" />
        <po:hasAction  rdf:resource="#DeliverBooks1" />
      </po:Rule>
    </po:hasRule>
    ....
  </po:hasRuleSet>
</sp:DeliveryPolicy>

<sp:CheckQuantity rdf:ID="CheckQuantity1">
  <po:hasExpression>
    <po:Expression rdf:ID="ExprCondition1">
      <po:hasVariable>
        <po:Attribute rdf:resource="sp:numberOfCopies" />
      </po:hasVariable>
```

```
      <po:hasOperator rdf:resource="po:isLessThanOrEqual" />
      <po:hasLiteralValue>
       <po:LiteralValue rdf:ID="LiteralValue1">
        <po:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int”>50
        </po:hasValue>
        <po:hasType rdf:resource="po:Integer" />
       </po:LiteralValue>
      </po:hasLiteralValue>
     </po:Expression>
    </po:hasExpression>
  </sp:CheckQuantity>

  <sp:DeliverBooks rdf:ID="DeliverBooks1">
   <po:associateWithOperation>
    <sp:DeliverBookOperation rdf:ID="DelBkStoreA" />
   </po:associateWithOperation>
   <po:hasExpression>
    <po:Expression rdf:ID="ExprAction1">
     <po:hasVariable>
      <po:Attribute rdf:resource="sp:deliveryDays" />
     </po:hasVariable>
     <po:hasOperator rdf:resource="po:isLessThanOrEqual" />
     <po:hasLiteralValue>
      <po:LiteralValue rdf:ID="LiteralValue2">
       <po:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int”>5
       </po:hasValue>
       <po:hasType rdf:resource="po:Integer" />
      </po:LiteralValue>
     </po:hasLiteralValue>
    </po:Expression>
   </po:hasExpression>
   <po:requireCondition rdf:resource="#CheckQuantity1" />
  </sp:DeliverBooks>
```

Fig.3. Example policy rule specification in OWL

2.2 QoS Properties

QoS is a very important aspect of non-functional properties for a service. In a distributed environment, services with equivalent functionality can be provided by different service providers with substantially varied QoS. How to specify QoS and incorporate it into service discovery process is thus of great importance.

QoS properties can be specified as *QoS parameters* and *QoS policies*. *QoS parameters* are QoS attributes that can be expressed in quantifiable measurements or metrics. A service usually possesses a set of QoS parameters. Though many of them are of dynamic nature, i.e., related to the execution of the service, a service can still advertise its guaranteed QoS in service description. *QoS policies* are rules related to QoS parameters. Rule-based QoS policies have often been used in network-related services, e.g. network and system management services. A service can provide different QoS classes of service depending on the service classes users subscribed. For example, a *GoldClass* user may have access to *GoldClassService*, which guarantees a set of QoS parameters, such as bandwidth and response time, much better than a user in *SilverClass*. A QoS policy for service discovery can similarly be specified as “If a user belongs to *GoldClass*, then the service provided guarantees a set of QoS parameters.”

QoS parameters can be classified into different categories, e.g., scalability, capacity, performance, reliability, availability, etc. There are several efforts to define and categorize QoS parameters in terms of classifications or ontologies [10][15]. Fig. 4 shows part of a QoS ontology based on [10] for illustration purpose.

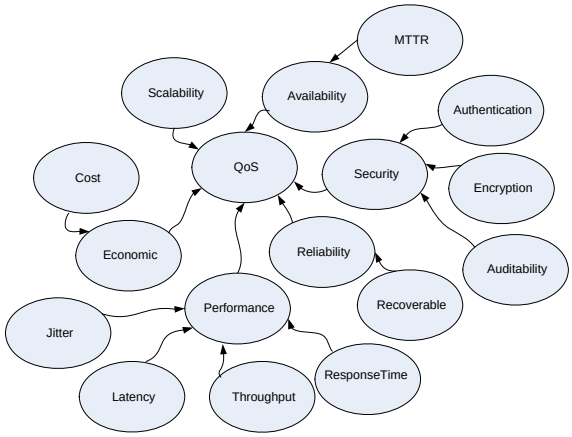


Fig. 4. Part of a QoS ontology (arrows indicate subClassOf relationship)

Not all attributes in the QoS ontology is relevant in a specific service discovery process, for example, a user may consider some of the QoS parameters valuable in his or her request. The matching procedure for service discovery therefore needs to take this into account by calculating the user specified QoS selection criteria. We adopt the QoS ranking approach proposed in [14], which defines a *quality matrix* to represent the values of user specified QoS parameters for all candidate services and an *overall QoS score function* to calculate overall QoS satisfactory values.

A *quality matrix*, $\Phi = \{V(Q_{ij});\ 1 \le i \le m;\ 1 \le j \le n\}$, is defined as a collection of quality attribute-values for a set of candidate services, where $V(Q_{ij})$ represents the value of the i^{th} QoS attribute for the j^{th} candidate service. These values are obtained from candidate service descriptions and mapped to a scale between 0 and 1.

An *overall QoS score function* is defined as

$$f_{QoS}(Service_j) = \sum_{i=1}^{m} (V(Q_{ij}) \times Weight_i)$$

where m is the number of QoS attributes in Φ, Weight_i is the weight value (specified by user) for each attribute.

The f_{QoS} score is calculated for each candidate service, and if the f_{QoS} score is greater than some user defined threshold, the corresponding service will be selected. Take an example, a user requesting an online streaming video service considers *throughput*, *response time* and *availability* more valuable than other QoS parameters and specifies the QoS selection criteria as $\text{Weight}_{Throughput} = 0.8$, $\text{Weight}_{responseTime} = 0.9$, $\text{Weight}_{Availability} = 0.7$, and a threshold score value $U_{Threshold} = 1.5$. Assume there are three candidate services for online streaming video, S_1, S_2 and S_3, and the quality matrix is:

$$\Phi = \begin{pmatrix} & S_1 & S_2 & S_3 \\ Throughput & 0.90 & 0.80 & 0.50 \\ responseTime & 0.90 & 0.80 & 0.60 \\ Availability & 0.90 & 0.50 & 0.40 \end{pmatrix}$$

After calculation of their respective f_{QoS} scores, only S_1 and S_2 will be selected. Further assume that the user specifies to rank the services based on *Cost* in ascending order, and the *Cost* of S_1 is greater than that of S_2, the results returning to the user will be $\{S_2, S_1\}$, specifying that S_2 is a better choice than S_1 for the user's purpose.

2.3 Context Policies

Context policies are rules related to context information. Some context information can greatly affect the selection of services. For instance, for a home food delivery service, the user's location is an important aspect for selecting possible service providers. In addition, depending on service types, different context information should be considered. Examples of context information include location, time, connection (e.g., if the user is accessible via a wireless or wired connection), user's feeling, presence, and user's habits and hobbies.

Context policies can be specified in the same format as business policies and QoS policies. For example, a service provider for a home food delivery service may specify a location-based policy as "only deliver food within the same city". This location policy can be specified as:

IF (*UserLocation.city* = *ServiceProviderLocation.city*)
THEN Service can be provided.

Some services are context-aware, others are not. For context-aware services, it is preferable that the context information can be automatically integrated into the service request even though the user does not specify them explicitly. For instance, there are several approaches to identify the location of a mobile user. One method to position the mobile user is to leverage the SS7 network to derive location. Another example is user profiles, which usually define user preferences, such as habits, hobbies, and other personalized information, such as access rights and startup applications. However, mechanisms for obtaining such context information are outside the scope of this paper. We just demand that context information should be included as a part of service request wherever possible so that context policies can be used during service discovery.

3. Integrated Semantic Service Discovery Framework

3.1 Integrated Semantic Service Description

There have been efforts to add semantics to service descriptions. Two major approaches based on ontology are OWL-S [3] and semantic-annotated WSDL [16]. OWL-S uses an OWL-based ontology for describing Web Services and supports service discovery at the semantic level. The semantic-annotated WSDL approach relates concepts in WSDL to OWL ontologies in Web Services descriptions, i.e., WSDL or UDDI. We adopt semantic-annotated WSDL to describe services, because WSDL has been accepted as

the industry standard for Web Services description and most of the existing Web Services support WSDL standards. This has the advantage of having widely acceptance without adding significant complexity.

Figure 5 demonstrates the semantic annotation for our integrated service description approach. An ontology-based semantic service description is represented as a semantic-annotated WSDL file, with links to the ontology definition and WS-Policy file for policy definitions and provided QoS parameters. This semantic-annotated WSDL is an XML-formatted Web Service description document based on WSDL, and is extended with OWL-based ontologies to add semantics to WSDL elements. The WSDL file *PurchaseBookService.wsdl* specifies the functional properties in terms of operations, inputs and outputs. Preonditions and effects for the operation can also be specified [17]. The concepts of them are referred to concepts in the service ontology. Policies for service discovery are specified in OWL file *PolicyRule.owl*, while WS-Policy framework and WS-PolicyAttachment are utilized to attach them to WSDL. In order to incorporate QoS parameters into WSDL without significant changes to existing WSDL structure, QoS parameters and other service parameters are also specified in an OWL file *QoS.owl*, and attached to WSDL using the same mechanism as the policy file. In detail, this means that all policies related to the service as well as QoS and service parameters can be specified in one XML file, *Policy.xml*, with links to respective OWL files, as shown in Fig. 6. This policy specification can then be attached to WSDL description, as shown in Fig. 7. As to be noted, we assume there is a shared ontology for each service domain, the same stands for policy and QoS ontologies. At the same time, a local ontology can be extended based on shared ontology to accommodate special needs.

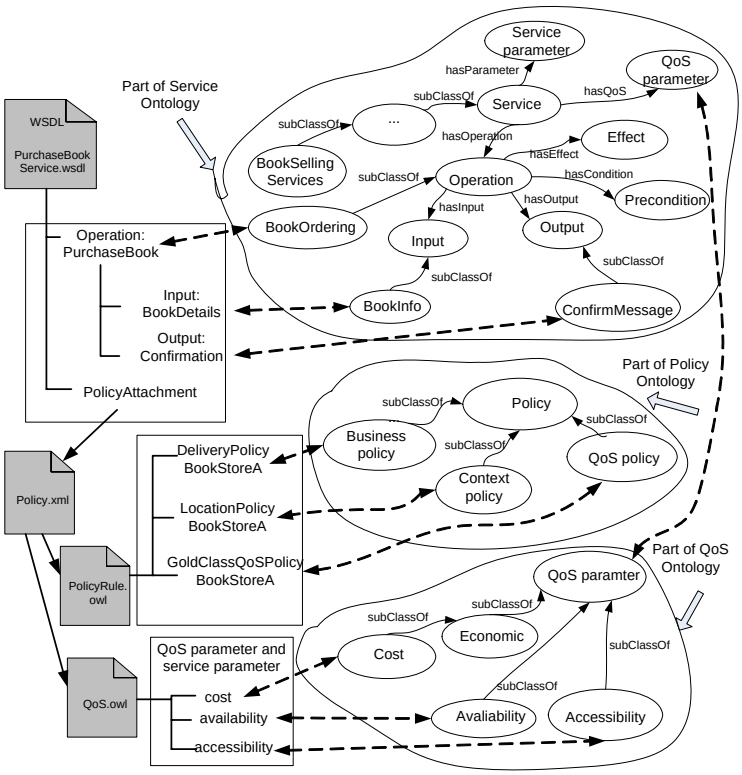


Fig. 5. Semantic annotated service description for integrated service discovery

```
<wsp:Policy Name="PurchaseBook">
  <wsp:All>
     <!- Business policy ->
     <po:DeliveryPolicy>http://BookStoreA.com/PolicyRule.owl#DelieveryPolicyBookStoreA
     </po:DeliveryPolicy>
     <!- Context policy ->
     <po:LocationPolicy>http://BookStoreA.com/PolicyRule.owl#LocationPolicyBookStoreA
    </po:LocationPolicy>
     <!- QoS policy ->
     <po:QoSPolicy>http://BookStoreA.com/PolicyRule.owl#GoldClassQoSPolicyBookStoreA
     </po:QoSPolicy>
      <!- QoS parameters and other service parameters ->
     <po:QoSParameters>http://BookStoreA.com/QoS.owl#QoSParametersBookStoreA
     </po:QoSParameters>
     <!- other policy for PurchaseBook, e.g. security policy ->
      ...
  </wsp:All>
</wsp:Policy>
```

Fig. 6. *Policy.xml* - All policy specification for Web Service *PurchaseBookService*

```
<wsp:PolicyAttachment>
  <wsp:AppliesTo>
    <wsa:EndpointReference>
      <wsa:ServiceName Name="PurchaseBookService" />
      <wsa:PortType Name="PurchaseBookPortType" />
      <wsa:Address URI="http://BookStoreA.com/PurchaseBookService" />
    </wsa:EndpointReference>
  </wsp:AppliesTo>
  <wsp:PolicyReference URI="http://BookStoreA.com/Policy.xml" />
</wsp:PolicyAttachment>
```

Fig. 7. Attaching policy specification to WSDL file *PurchaseBookService.wsdl*

3.2 Integrated Semantic Service Requirement

A user request specifies the functional requirements, non-functional requirements and user defined selection criteria in terms of preferred QoS parameters and their weights. Such request is also based on ontology concepts. A request template can be provided. This user request is combined with automatically obtained context information to produce an integrated service requirement specification in the form of an *integrated semantic service request*, which comprises the following information:

- *Functional requirements* in terms of operations, inputs, outputs, preconditions and effects.
- *Non-functional requirements,* such as QoS constraints.
- *Context information* obtained automatically by the system.
- *User specified selection criteria,* i.e. QoS parameters and their weights as well as user specified ranking criteria. This allows for personalized service ranking.

3.3 Integrated Semantic Service Discovery Procedure

Integrated semantic service discovery process can be arranged in two major steps. The first step is to find out the services that meet the functional requirements based on matching of functional properties. As there is usually more than one service matching

the functional requirements, a set of candidate services are obtained. The next step is therefore to select the most appropriate ones from these candidates based on non-functional properties and rank them according to user defined criteria.

The whole discovery process is carried out by a reasoning engine. When an integrated semantic service request is sent to the reasoning engine, the engine will first determine the candidate services that offer the requested functionality based on matching of functional properties. We adopt a procedure based on ontological inference and degree of match [13]. This procedure typically uses subsumption reasoning to find similarity between service descriptions and service requests based on operations, inputs and outputs. Preconditions and effects can also be used for matching. During the second step, policies will be checked and applied to further select services among the candidate services. The semantic-annotated WSDL files of candidate services contain links to all policy specification file (e.g. *Policy.xml*), which can be referenced to retrieve the related policy rules (*PolicyRule.owl*) as well as QoS and service parameters (*QoS.owl*). Rule-based reasoning can then be applied to determine satisfied matching. After that, overall QoS scores for those candidate services are calculated based on user defined selection criteria (i.e. based on selected QoS parameters and their respective weights) as described in Sect. 2.2. All the matches will be returned according to user specified ranking criteria.

Ontological inference and rule-based reasoning are applied during semantic service discovery process. The reasoning engine which carries out the above procedure is based on XDD [23] – a knowledge representation framework - and XET [2] – a powerful computing and reasoning engine for XDD. Work has already been done, based on this representation framework and reasoning engine, for dynamic service configuration [1], composition [20], and management [9], proving the practicability and reasoning power of such reasoning engine.

XDD (XML Declarative Description) is an expressive XML rule-based knowledge representation, which extends ordinary, well-formed XML elements by incorporation of variables for an enhancement of expressive power and representation of information into so called *XML expressions*. A description in XDD is a set of XML expressions and the XML elements' relationships in terms of *XML clauses*. XML expressions represent facts, while XML clauses express rules, conditional relationships, constraints and ontological axioms. Applying XDD framework, Ontology-annotated WSDL descriptions, concepts and properties in OWL-based ontologies, service parameters and QoS parameters can all be represented as facts using XML unit clauses. Ontological relations and axioms as well as policy rules for service discovery can be represented as rules using XML non-unit clauses. Rules can also be defined for ontological inference and querying in XDD. Service requests can be represented as XDD query clauses using XML clauses, which specify the patterns as well as the selection conditions of the queries. This means all information and rules for integrated semantic service discovery can be directly represented as XDD descriptions. Furthermore, XDD descriptions can be computed and reasoned using XET (XML Equivalent Transformation), a Java-based reasoning engine that transforms the query clause by the XDD-based rules based on equivalent transformation [2]. Therefore, by expressing ontologies and rules directly in XDD and executing service discovery queries using XET-based engines, we eliminate the overhead of transforming between ontology language and rule-based representation,

and can achieve both ontological inference and rule-based reasoning, two fundamental functionalities for semantic service discovery process.

4. Related Work

Several approaches for ontology-based semantic service discovery have been proposed, based on OWL-S [13] or semantic-annotated WSDL [16]. However, both of them only apply ontology for matching on the operational interfaces (i.e. input and output parameters of the operations of the Web Services). In addition, both of them lack mechanisms to represent non-functional properties based on rule-based policies. We extend the semantic matching and selection based on non-functional properties, i.e. ontology-based policy rules and QoS parameters.

Sriharee et al. [18] proposed to discover Web Services based on business rules policy using WS-Policy and ontology, but without further consideration of QoS attributes. For incorporating QoS attributes with service discovery, Zhou et al. [24] proposed a DAML-QoS ontology for specifying various QoS properties and metrics. However, there was no provision for the users to specify ranking criteria (based on non-functional properties) for service selection. The framework proposed by Pathak et al. [14] provides QoS-based service selection; however, there was no consideration for policy rules during the discovery process. Maximilien et al. [10] proposed a framework and ontology for service selection also considering QoS properties, but there was no provision for user-specified ranking criteria in service request.

5. Conclusions

An approach to integrated semantic service discovery is presented. We first describe how non-functional properties are expressed based on business policies, QoS policies and context policies as well as QoS parameters. We then present our approach for adding semantics to service description for both functional and non-functional properties based on ontologies. We further show how service request can be integrated with context information and personalized ranking criteria. Based on them, an integrated semantic service discovery procedure based on both functional and non-functional properties is presented.

We base our work on widely accepted standards in Web Services and Semantic Web, i.e., WSDL, OWL and WS-Policy. The integrated semantic service discovery approach is a rather generic one, and can be applied in a centralized or distributed environment. We are working towards mechanisms to apply this approach to autonomic environments with distributed, self-organizing and scale-free communications. Issues about how the service descriptions are stored and organized as well as how they are accessed need further exploration.

Shared ontologies are assumed for service descriptions and service requests in our approach. If different ontologies are used, ontology mapping should be carried out to build up correspondence between ontologies used for service descriptions and those used for service requests.

References

1 F. A. Aagesen, P. Supadulchai, C. Anutariya, and M. M. Shiaa. Configuration management for an adaptable service system. In *IFIP Int'l Conference on Metropolitan Area Networks, Architecture, Protocols, Control and Management, proceedings*, Ho ChiMinh City, VietNam, 2005.

2 C. Anutariya, V. Wuwongse, and V. Wattanapailin. An equivalent-transformationbased xml rule language. In *Int'l Workshop Rule Markup Languages for Business Rules in the Semantic Web, proceedings*, Sardinia, Italy, 2002.

3 The OWL Services Coalition. Owl-s: Semantic markup for web services, 2003. http://www.daml.org/services/owl-s/1.0/owl-s.html.

4 The Salutation Consortium. Salutation architecture specification version 2.0c, 1999. http://www.salutation.org/.

5 S. Bajaj et al. Web services policy attachment, 2006. http://www-128.ibm.com/developerworks/library/specification/ws-polatt/ .

6 S. Bajaj et al. Web services policy framework (ws-policy), 2006. http://www-128.ibm.com/developerworks/library/specification/ws-polfram/.

7 UPnP Forum. Upnp device architecture version 1.0, 2000. http://www.upnp.org/.

8 E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. RFC2608, 1999.

9 S. Jiang, M. M. Shiaa, and F. A. Aagesen. An approach for dynamic service management. In *EUNICE'04, Proceedings*, Tampere, Finland, 2004.

10 E. M. Maximilien and M. P. Singh. A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, 8(5):84–93, 2004.

11 Sun Microsystems. Jini architecture specification version 2.0, 2003. http://www.jini.org/.

12 OWL. Owl web ontology language overview. W3C Recommendation, Feb 2004. http://www.w3.org/TR/owl-features/.

13 M. Paolucci, T. Kawmura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *First Int. Semantic Web Conf., Proceedings*, 2002.

14 J. Pathak, N. Koul, D. Caragea, and V. Honavar. A framework for semantic web services discovery. In *WIDM'05, Proceedings*, 2005.

15 S. Ran. A model for web services discovery with qos. *ACM SIGecom Exchanges*, 4(1):1–10, 2003.

16 K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding semantics to web services standards. In *ICWS'03, Proceedings*, 2003.

17 N. Sriharee and T. Senivongse. Discovering web services using behavioural constraints and ontology. In *DAIS'03*, volume 2893 of *LNCS*, pages 248–259. Springer, 2003.

18 N. Sriharee, T. Senivongse, K. Verma, and S. Sheth. On using ws-policy, ontology, and rule reasoning to discover web services. In *INTELLCOMM 2004, Proceedings*, volume 3283 of *LNCS*, pages 246–255, Bangkok, Thailand, 2004. Springer.

19 R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, 25(1-2):161–197, 1998.

20 P. Supadulchai and F. A. Aagesen. A framework for dynamic service composition. In *First Int'l IEEE Workshop on Autonomic Communications and Computing, Proceedings*, Taormina, Italy, 2005.

21 uddi.org. Universal description, discovery and integration of web services. http://www.uddi.org/.
22 W3C. Web services description language (wsdl)1.1, 2001. http://www.w3.org/TR/wsdl.
23 V. Wuwongse, C. Anutariya, K. Akama, and E. Natajeewarawat. Xml declarative description: A language for the semantic web. *IEEE Intelligent Systems*, 16(3):54–65, 2001.
24 C. Zhou, L. Chia, and B. Lee. Service discovery and measurement based on daml-qos ontology. In *Special Interest Tracks and Posters of 14th World Wide Web Conference, Proceedings*, 2005.

PAPER B: A Self-organizing Service Discovery System Based on Semantic Overlay Networks

Shanshan Jiang, Finn Arve Aagesen and Hao Ding

Published in
Journal of System and Information Sciences Notes, July 2007
Volume 1, Number 3, pp. 303-309.
@SIWN 2007

SIWN International Conference on Complex Open Distributed Systems(CODS 2007)
Chengdu, China, July 22-24, 2007.

A Self-organizing Service Discovery System Based on Semantic Overlay Networks

Shanshan Jiang

Finn Arve Aagesen

Hao Ding

Abstract A network of autonomous directories forming a large-scale distributed service discovery system is considered. It is proposed to organize directories into Semantic Overlay Networks (SON) based on service ontology. The various SONs are further organized in super-peer networking structures. The focus is on the functionality for the organization of directories into SONs and the self-organizing construction and maintenance of super-peer SON networks. Simulations indicate that the proposed functionality gives super-peer networks with a small number of SONs and a small SON size that require a small management procedure overhead. The self-organization time is short both for SON initial construction and in node leaving situations. Moreover, discovery procedure overhead is significantly reduced compared to a system based on a random overlay network.

Keywords: self-organizing, semantic overlay network, service discovery, super-peer network.

1. Introduction

In a distributed service environment, service discovery is a core functionality to locate desired services. *Service discovery* is the process of finding the desired services by matching *service descriptions* against *service requests*. A *service description* provides service-related information which can be advertised by a service provider and searched during service discovery process. A *service request* represents user's service requirements. Both service descriptions and requests comprise information on functional and non-functional properties. Ontologies are the basis for adding semantic expressiveness to service descriptions and requests. An *ontology* is an explicit and formal specification of a shared conceptualization [14]. A *service ontology* is accordingly an explicit and formal specification of core concepts of the functional and non-functional properties of service. *Semantic service discovery* is a service discovery process based on ontology concepts. Likewise, *semantic matching* is the matching of service requests and service descriptions based on ontology concepts.

Available approaches for service discovery, such as SLP [4], Jini [15] and UPnP [17], are targeted for enterprise and home networks. They either rely on centralized entities for discovery or search entire local network by broadcast or multicast. However, the location of services on Internet and other large-scale environments with a huge amount of a large variety of services needs other solutions. Hence, an important

research objective is the location of services in such large-scale systems efficiently, *i.e. to answer service requests fast with low overhead.*

We consider a large-scale service discovery system which consists of autonomous directories, where each directory has its own local registered service descriptions. We propose to apply semantic service discovery and to organize directories into *Semantic Overlay Networks* (SON). SON is a flexible network organization that logically connects nodes with semantically similar contents. The concept of SON is introduced in [2], which aims to improve query performance while maintaining a high degree of node autonomy. In this paper, a super-peer [19] based networking architecture is proposed for the various SONs. A super-peer architecture organizes nodes into hierarchy by utilizing the different capacities of nodes. Super-peers are peers with high capacity and provide services to their clients. They are dynamically selected by some super-peer selection algorithm and do accordingly not constitute single points of failure. A super-peer based approach thus has the efficiency of a centralized approach and the scalability, load-balancing and robustness of a distributed approach.

For the organization of directories into SONs, a shared service ontology based on a predefined service category hierarchy is used. The service descriptions are classified into service categories based on service ontology, and directories with semantically similar descriptions (i.e. similar service categories) are grouped into SONs. Each service request also identifies one service category.

Service discovery based on SONs is an iterative process that can consist of several loops. Each loop has two steps. In Step 1, SONs which may contain relevant directories are selected. In Step 2, all the directories in these SONs are searched based on semantic matching. In each loop, the results are checked to see if enough matches are found. If not, more SONs are selected in new loops and until enough matches are obtained.

Efficient service discovery requires that *the service requests can be answered fast* and that *the overhead for successful search is low*. This requires that the system should *enable high probability searching* (Requirement **R1**). This requirement means that the number of SONs selected for answering a request in Step 1 should be small and at the same time contain directories that have a high number of matches. Moreover, to reduce the maintenance overhead as well as the messages for answering a request, it also requires that *the number of SONs as well as the size of SONs must be small* (Requirement **R2**).

The focus of this paper is on Step 1, i.e. how to select a relatively small set of promising directories out of a large number of directories. In a previous work [7], an integrated semantic service discovery approach that can carry out semantic matching has been proposed for Step 2.

The rest of the paper is organized as follows: Sect. 2 discusses related work. Sect. 3 presents a SON-based service system model, while Sect. 4 describes the SON-based service discovery system. Experiments and results are described in Sect. 5, followed by conclusions in Sect. 6.

2. Related Work

Semantic Overlay Networks have been proposed as an effective way to improve search in Peer-to-Peer (P2P) systems. The concept was originally introduced in [2], which also defined the challenges for building SONs as follows: 1) classification of queries and

peers, 2) definition of level of granularity for each classification, 3) conditions for a peer to join a SON and d) the selection of SONs for answering a query. The actual networking aspect of SONs is however not addressed. Lately several approaches have been proposed. For example, SONs based on unstructured P2P systems (i.e. random overlay networks) have been proposed in [18][16]. However, unstructured P2P systems usually relying on flooding for searching are not efficient for large-scale service discovery. DHT (Distributed Hash Table)-based P2P systems [12][13] can locate nodes with a small number of messages based on DHT algorithms. They can however not process complex queries which are important for semantic service discovery. In addition, the high maintenance cost for DHT-based index and tight coupling between nodes limits their application for efficient wide area service discovery.

Super-peer architecture has been proposed for providing the efficiency of a centralized approach and the scalability, load-balancing and robustness of a distributed approach. In [8], clustering policies are proposed to generate semantic clusters in super-peer networks. Our approach differs by clustering directories based on service category hierarchy. Furthermore, our approach considers the self-organizing and adaptation of SONs, where the number of SONs and SON membership can change dynamically by applying adaptation policies on each directory. In addition, the super-peers in our system do not index data of their clients and are mainly used to route requests to proper SONs.

This paper adopts a modified version of SG-1 [9] for super-peer selection. A recent protocol and a natural evolution of the SG-1 algorithm is SG-2 [6], which introduces the notion of latency between peers and poses a QoS limit on it. In addition, SG-2 is strongly bio-inspired.

The self-organizing process for construction and maintenance of SONs in this paper is based on gossiping [3]. The gossiping paradigm has been applied in self-organizing environments for other purposes as well. For example, gossip-based clock synchronization for large decentralized systems is proposed in [5].

3. SON-based Service Discovery System Model

A *service discovery system* Ψ can be modeled as $\Psi = <O, D, L, F>$, where O is the *service ontology*, D is a set of directories, L is a set of *links* that logically connects directories based on semantic similarity and F is a set of *functionality*.

The service ontology O defines the service categories as well as other service related concepts in a hierarchical structure. The upper ontology of service *s* is defined as a tuple <*c, p, op, qos, sp*>, where *c* denotes the *service category* the service *s* belongs to, *p* the *policies* applied to *s*, *op* the *operations s* performs, *qos* the *QoS parameters* and *sp* the *service parameters*. As shown in Fig. 1, a *service* belongs to a *service category*, and has *operations, policies, service parameters* and *QoS parameters*. Each operation is defined by *inputs, outputs, preconditions* and *effects*. The set of categories $\{c\}$ is connected by a rooted tree, the *category hierarchy* $\mathcal{CH}$. The role and the use of the service categories is explained at the end of this Section. More details of definitions of service ontology elements and their application in semantic service discovery can be found in our previous work [7].

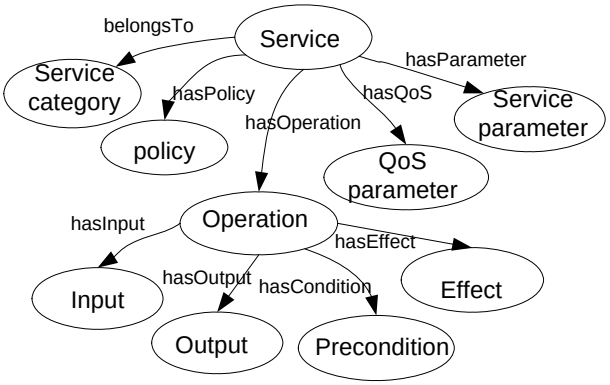


Fig. 1. Upper ontology of service.

$\mathbb{D}$ is the set of directories $\{d_i\}$. Each d_i maintains a set of *service descriptions* $SD_i = \{sd_j\}$. Each service description sd_j belongs to one service category c, denoted as $sd_j \rightarrow c$. Each d_i is *logically* connected with a relatively small set of directories, called its *neighbors* N_i.

$\mathbb{L}$ is a set of *links* $\{l\}$ that logically connects directories based on semantic similarity. A *link* l is a triple $<d_i, d_j, c>$ where d_i and d_j are the logically connected directories based on a service category c. A link is symmetric, i.e., $<d_i, d_j, c>$ is the same as $<d_j, d_i, c>$. A *Semantic Overlay Network (SON)* is represented by the set of links with the same c, $SON_c = \{d_i, d_j \in \mathbb{D} \mid . \exists$ a link $l = <d_i, d_j, c>\}$. Each directory can join one or several SONs.

$\mathbb{F}$ is a set of *functionality* provided by the system, which includes six operations as described as follows. Each SON_c supports three operations:

- $Join_c(d_i, c)$: a directory d_i is added to SON_c, which means to find a directory $d_j \in SON_c$ so that a link $<d_i, d_j, c>$ can be added to the system Ψ.
- $Search_c(r, c)$: the service request r is sent to all the directories in SON_c and a set of matches are returned.
- $Leave_c(d_i, c)$: a directory d_i leaves SON_c by simply dropping all the links d_i maintains.

In addition to the above three operations related to SONs, the service discovery system also supports the following operations:

- *Register*(sd_j, d_i, c): a service description sd_j is registered in a directory d_i with service category c.
- *Assign*(d_i): a directory d_i is assigned to a number of SONs according to the service descriptions registered.
- *Discovery*(r, c): a service request r with a service category c specified by the user is sent to the system and routed to relevant directories for discovery.

The role and the use of the service categories can be explained as follows. In service ontology, the category hierarchy $\mathcal{CH}$ is defined and is referenced by both users and service providers when they define service requests and descriptions. Each service description is mapped to one service category when it joins the system. Each service request contains one service category c_r which defines the range of services that the user is looking for. Each SON_c is related to one service category c, which is the root of the

category subtree from $\mathcal{CH}$. For example, in Fig. 2, the subtree of c_2' consists of {c_3, c_4, c_2'}. Each directory has a set of service categories {c} from all the service descriptions registered in the directory. By aggregation of service descriptions according to their semantic relationships, a new set of categories {c'} is obtained (cf. Sect. 4.1). Each c' corresponds to one SON_c' that the directory is member of. A directory joining SON_c' implies that it has enough service descriptions that fall under the subtree of c'. Fig. 2 illustrates the category relations in a directory. The black circles represent categories related to service descriptions, while larger circles represent categories related to SONs.

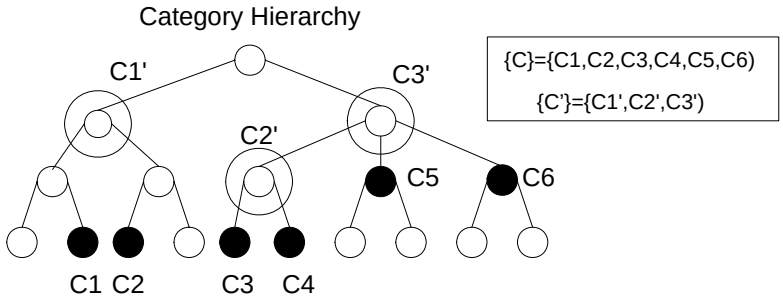


Fig. 2. Example of category sets of a directory.

4. SON-based Service Discovery System

The architecture for SON-based service discovery system is illustrated in Fig. 3. Directories are logically organized into SONs. Each directory can belong to several SONs. Each SON has an *entry directory*, which is a directory with high capacity. The entry directory acts as a *super-peer* for other client directories in a SON. A client sends requests and receives answers via its super-peer. In addition, such super-peers are connected with each other for inter-SON communication. In other words, the entry directory (super-peer) can distribute messages within a SON as well as forward them to different SONs for global discovery. The inter-SON communication is basically broadcast-based, although other approaches can be applied, such as DHT [13]. Each directory has the following functionality:

- Registration of service descriptions.
- Assigning the directory to SONs, maintaining SON connections and adapting to dynamic changes.
- Accepting service requests and carrying out local semantic matching procedure.

In addition, two more functionalities are needed, i.e. for:

- SON super-peer network construction.
- SON super-peer network maintenance, including directory joining and leaving.

Each service is registered in a local directory with a service category. Policies are applied when assigning directories to SONs and mapping service request to SONs for discovery. Service discovery requires also the iterative selection of SONs and searching in each SON. In the following, assignment of directories to SONs, construction and maintenance of SONs as well as service discovery are explained in more detail.

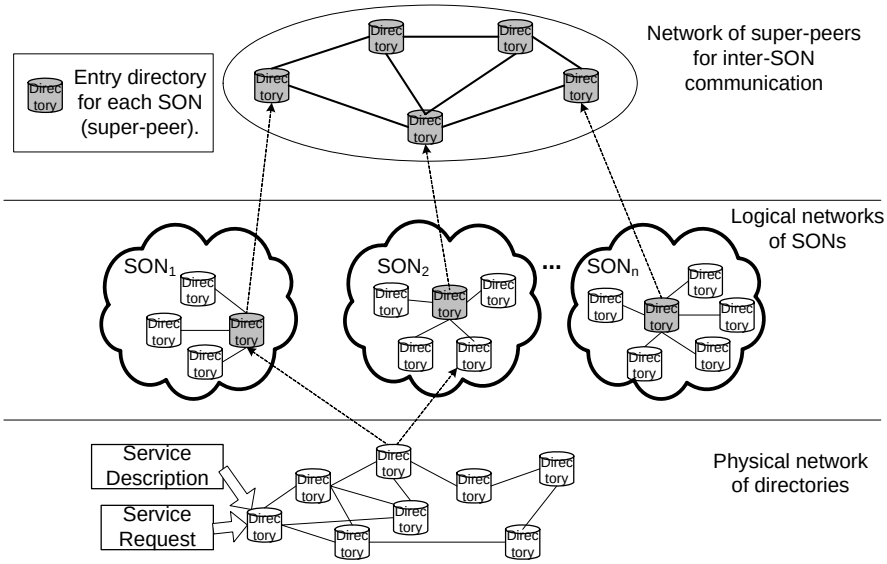


Fig. 3. SON-based service discovery system architecture.

4.1 Assignment of Directories to SONs

To join the SON-based service discovery system, a directory needs to find out which SONs to join based on the service categories of the service descriptions contained in it. The procedure for determining which SONs to join consists of two steps: 1) aggregating service descriptions into groups, and 2) assigning directories to SONs according to assignment policies based on such group information. In addition, as services can dynamically join and leave the system, the services registered in the directories can change, which may cause SON membership to change. Adaptation policies are therefore accordingly applied for directories to adapt to runtime situation.

4.1.1 Aggregation Policy

To meet the requirement **R2**, there should be aggregation of service categories of all service descriptions stored in a directory. This is based on the semantic relationships between service categories. For two service categories *c* and *c'*, $c' \preceq c$ means *c'* equals to *c* or *c'* is a descendant of *c* in the service category hierarchy. To aggregate service descriptions, a service description *sd* will be placed in a group of service category *c* (denoted as $sd \Rightarrow c$) if ($sd \rightarrow c'$) and ($c' \preceq c$). In other words, *c* is the root of subtree that *c'* belongs to.

4.1.2 Assignment Policies

Assignment of directories to SONs is based on group information according to assignment policies. Several assignment policies can be defined.

- *Policy A1*: a directory d_i joins a SON_c if it has a service description that belongs to group *c*, i.e. ($d_i \in SON_c$) if ($\exists\ sd_j \Rightarrow c$, $sd_j \in d_i$).

This is the most conservative policy. However, such policy tends to produce too many links.

- *Policy A2*: a directory d_i joins a SON_c if the number of service descriptions that belongs to *c* is greater than a threshold T, i.e. ($d_i \in SON_c$) if ($Count(sd_j \Rightarrow c) > T$).

Such policy is good and practical for service discovery in Internet-scale environment. If a directory d_i joins a SON_c, this implies that d_i contains many service descriptions falling under the subtree of *c*; therefore, selecting d_i for matching will provide higher number of matches than selecting the directories that are not members of SON_c. Policy A2 thus meets the *high probability searching* requirement (**R1**). The threshold value T can be an adjustable parameter, which is important for the performance of SON-based service discovery system. Taking a high T value, SON connections per directory can be reduced, and accordingly SON sizes are reduced (**R2**). For example, when the threshold parameter T=10, a node will only join SON_c when it has more than 10 service descriptions belonging to group *c*. The SON connections per node and SON sizes will accordingly be changed by the variation of T. However, for small groups of service descriptions that are not assigned to SONs, there will be possibility that such descriptions never be searched if they are not associated with any SON. To compensate, Policy A3 can be applied.

- *Policy A3*: for service groups that do not have enough service descriptions, the directory can join a SON that corresponds to an ancestor of current service category.

The selection of assignment policies is determined by the distribution of services. If services are clustered, each directory contains only a few service category groups. Both the number and the size of SONs can be small if applying Policy A1. On the other hand, if services are evenly distributed, Policy A2 and A3 need to be applied to reduce the SON connections. In this case, services registered in a directory that contains only a few instances of the same category may not be searched if other directories contain many instances of the same category due to high probability searching. However, this is acceptable for large-scale service discovery since users usually demand *any* service instance that can meet their requests. Otherwise, iterative selection of SONs (cf. Sect. 4.3.3) will search each possible service in the expense of higher number of messages.

4.1.3 Adaptation Policies

The following adaptation policies are defined for directories to adapt to runtime situation:

- *Policy P1*: the number of SONs a directory joins must be less than a threshold value *Ts*. Such *Ts* is determined by the directory's capacity.
- *Policy P2*: if the number of service descriptions belonging to the subtree of *c* is over a threshold value *Tc* and there is no connection to SON_c yet, then add the directory to SON_c.
- *Policy P3*: if the number of service descriptions belonging to the subtree of *c* drops below *Tc*, the directory can be removed from SON_c, but must make sure that it is still connected to one SON that corresponds to *c*'s ancestor.

4.2 Construction and Maintenance of SONs

Self-organization means the system can dynamically adapt to topology change and reorganize each node's neighbors. A self-organizing process is applied for both construction and maintenance of SONs based on gossiping [3]. It is an autonomous functionality determined by directories themselves. By periodically exchanging current

status information (called *partial view*) with randomly selected peer nodes, old information gradually and automatically replaced by new information. Network topology can thus be automatically updated. In particular, information about failed nodes will be automatically removed from the system, allowing the system to "self-repair" the overlay topology.

4.2.1 Construction

Initially, all the directories in the system decide which SONs to join according to the function described in Sect. 4.1. For each SON, one directory with high capacity will be selected as SON entry directory. Some super-peer selection algorithms can be applied. In this paper, a modified version of SG-1 algorithm [9] is adopted. This algorithm is based on gossiping and assures the dynamic selection of super-peers according to runtime situation. Each node periodically exchanges its *partial view* with randomly selected peer nodes, which contains information such as identifier, capacity, current role (super-peer or client), SON memberships, the number of clients they are serving and neighbors. After exchanging views, nodes with available connections and higher capacity will be changed into super-peers, and others are to connect to them as clients. Alternatively, a super-peer may decide to move all its clients to another super-peer with more capacity, and become a client itself. This role changing is particular useful when the super-peer is overloaded or degraded, so that load balancing can be achieved. Such process continues until the system becomes stable, i.e., the minimum number of super-peers is selected.

4.2.2 Maintenance

The maintenance of SONs involves the handling of the following situations:

- *Directory joining*: when a new directory decides to join a SON (according to the function defined in Sect. 4.1), it contacts the entry directory for the SON and joins the SON as a client of the entry directory. If it is the first member or the entry directory has no capacity to accept more clients, it can declare itself as an entry directory (super-peer), and wait to accept other clients.
- *Directory leaving*: if the directory is a client, it does not need to do anything. Such change will be automatically updated by gossiping. However, if it is a super-peer, all its clients need to be connected to new super-peers. In the system, a client periodically probes its super-peer to see if it is active. If its super-peer fails or leaves, the client can declare itself as a super-peer, and participate in the super-peer selection process as described in Sect. 4.2.1.
- *SON membership update*: each directory can determine to update its SON membership according to registered services by applying adaptation policies defined in Sect. 4.1.3. The update process can be viewed as consisting of directory leaving and directory joining.

4.3 Service Discovery

When a request is sent to the service discovery system, the discovery operation *Discovery*(r, *c*) is carried out by an iterative procedure consisting of two steps in each loop.

4.3.1 Step 1: Select the Relevant SONs

Given a service request with category c, the system needs to map it to SONs which may contain relevant service descriptions. The selection of relevant SONs is determined by the semantic distance between c and c_s (category for a candidate SON) in the $\mathcal{CH}$. Different selection policies give different discovery scope:

- *Policy S1*: Only SON_c is selected.
- *Policy S2*: Select SONs associated with c and its descendents c'.
- *Policy S3*: Select SONs associated with c, and both its descendants and ancestors c'.

An iterative selection based on *Policy S3* is adopted in our system (described in Sect. 4.3.3).

4.3.2 Step 2: Search Each Directory in SON_c

This corresponds to the operation $Search_c(r, c)$. Since each SON is managed by a super-peer with its clients, the super-peer will forward the service request to each directory in the SON. Then semantic matching is carried out in each directory. This is based on other service concepts than service category in the service ontology, such as operations, service and QoS parameters. The semantic matching is based on semantic similarity between ontology concepts. The results of a semantic service discovery can be ranked according to functionality similarity based on degree of match, which can be distinguished as *Exact, Plugin, Subsume* and *Fail* [10]. A match between a service description *sd* and a service request *r* can then be modeled as a function $M(sd, r) = \{1, \text{if } degree \in (Exact, Plugin, Subsume) \mid 0, \text{if } Fail\}$. Our previous paper [7] proposed an approach for semantic matching based on service ontology defined in Fig. 1. As this is not the focus of this paper, interested reader should refer to [7].

4.3.3 Iterative Selection of SONs

To improve discovery efficiency and enable *high probability searching* (**R1**), SONs with most probability to answer the request are selected first to carry out Step 2 mentioned above, then less probable SONs are selected until enough matches are obtained. In detail, given a request for category c, SON_c (if it exists) is searched first. If not enough matches are obtained, SONs that correspond to c's descendents (if any) are searched. If still not enough matches returned, SONs that correspond to c's ancestors are searched (if any, and from the nearest ancestor until the root of the $\mathcal{CH}$). Such process continues until enough matches are obtained or every possible SONs are searched. Since the size of each SON is determined by the capacity of its entry directory, there may be several SONs corresponding to one category c. In this case, each such SON is selected until enough results are obtained.

5. Evaluation

To evaluate the system performance, the following quality measures are used:

- *number of SONs and SON size*
- *management procedure overhead*

- *discovery procedure overhead*
- *self-organization time*

The number of messages sent per directory in order to maintain super-peer based SON structure is applied as a measure for *management procedure overhead. Discovery procedure overhead* is the number of messages generated by the discovery system to answer a service request.

A simulation round δ is a *logical* time unit, representing the interval each node exchanges messages with its neighbor (gossiping interval). As a measure for *self-organization time*, the number of simulation rounds for the system to stabilize is used. The number of rounds corresponds to how many times a node exchanges messages. The real time for execution is determined by the actual gossiping interval. For example, if the gossiping interval is 1s, then the system can stabilize in about 10*δ = 10s. A smaller number of rounds means the system can stabilize faster.

An efficient SON-based service discovery system requires *a small SON size and a small number of SONs, a small management procedure overhead value, a small discovery procedure overhead value and a short self-organization time.*

Simulations based on the PeerSim framework [11] are carried out to investigate the proposed system. To realize the $Search_c(r, c)$ and $Join_c(d_i, c)$ operations, a general super-peer based protocol for communication within individual SON is implemented and an existing implementation of Newscast -- a gossip-based membership protocol [11] for inter-SON communication is used. SONs are constructed according to the procedure defined in Sect. 4.2.1. Power-law distribution on directory capacities are applied, which also determines SON sizes. Assume each directory belongs to one SON and only one SON is selected for each request. Simulation rounds for various network sizes are conducted, i.e. from 1000 to 50000 nodes (directories). To evaluate the self-organizing property of the SON system, 50 nodes are deleted in each round from round 30 to round 35.

The SON size is determined by the capacity of its super-peer. The number of SON connections per directory is determined by the policies applied when assigning directories to SONs (cf. Sect. 4.1). In the simulation, one SON per directory is applied. Table 1 shows the number of SONs and the average SON size when the system stabilizes. The results indicate that the number of SONs in the system is small. If a directory joins more than one SON, the number of SONs may increase.

Table 1. The number of SONs and average SON size for different network sizes.

Number of Nodes	1000	5000	10000	50000
Number of SONs	10	44	82	396
Average SON size	100	114	122	126

Fig. 4 shows that our proposed SON-based system becomes stable in approximately 10 rounds in both SONs construction situation and node leaving situation. Fig. 5 shows the average management procedure overhead in each round. Generally, the overhead is between 1 and 6. Fig. 6 shows the discovery procedure overhead for different network sizes. The results of SON-based systems are compared with searching in random overlay network-based systems, such as Gnutella-based searching [1]. As SON-based

system routes the requests only to a subset of relative directories, the discovery procedure overhead is significantly reduced.

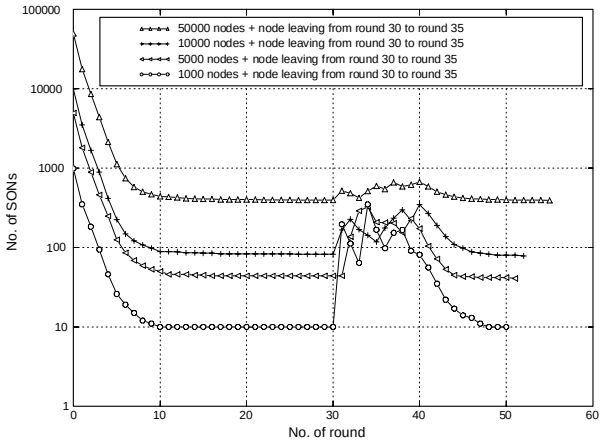


Fig. 4. The number of SONs and self-organization time for different network sizes.

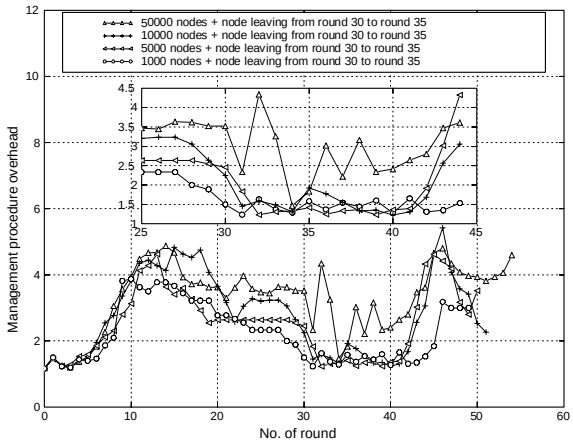


Fig. 5. The average management procedure overhead for different network sizes.

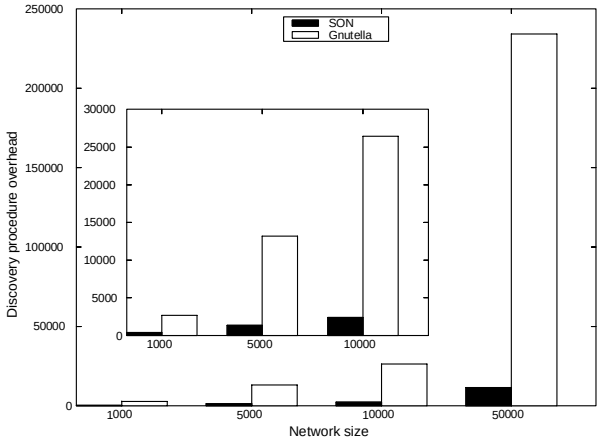


Fig. 6. Discovery procedure overhead for different network sizes.

6. Conclusion

Semantic Overlay Networks (SONs) have been proposed for improving the discovery efficiency, i.e. to answer service requests fast with low overhead, in large-scale service discovery systems. The focus is on functionality for the organization of directories into SONs and the self-organizing construction and maintenance of super-peer SON networks. Different policies are applied as constraints on the system functionality. Aggregation and assignment policies are applied to ensure the size and number of SONs can be small. The number of SONs and SON membership can change dynamically by applying adaptation policies on each directory. Iterative selection of SONs for discovery enables high probability searching. The proposed system has been evaluated by simulations. The results indicate that such super-peer based SON system can improve discovery efficiency. The proposed functionality gives a small number of SONs with small SON size and requires a small management procedure overhead. The results also indicate short self-organization time both in initial SON construction situations and in situations when nodes are leaving. Moreover, discovery procedure overhead is significantly reduced compared to a system based on a random overlay network.

References

[1] Clip2 distributed Search Services, The Gnutella protocol specification v0.4.

[2] A Crespo and H Garcia-Molina, Semantic overlay networks for P2P systems. Technical Report, Computer Science Department, Stanford University, October 2002.

[3] A Demers, et al, Epidemic algorithms for replicated database maintenance. Proceedings of the sixth annual ACM Symposium on Principles of distributed computing (POCD 1987), Vancouver, British Columbia, Canada, August 10 - 12, 1987.

[4] E Guttman, C Perkins, J Veizades and M Day, Service location protocol, version 2. RFC2608, 1999.

[5] K Iwanicki, M van Steen and S Voulgaris, Gossip-based clock synchronization for large decentralized systems. Proceedings of the Second IEEE International Workshop on Self-Managed Networks, Systems and Services (SelfMan 2006), Dublin, Ireland, June 16, 2006, LNCS 3996, pp. 28-42.

[6] G P Jesi, A Montresor and O Babaoglu, Proximity-aware superpeer overlay topologies. Proceedings of the Second IEEE International Workshop on Self-Managed Networks, Systems and Services (SelfMan 2006), Dublin, Ireland, June 16, 2006, LNCS 3996, pp. 43-57.

[7] S Jiang and F A Aagesen, An approach to integrated semantic service discovery. Proceedings of the First International IFIP TC6 Conference on Autonomic Networking (AN 2006), Paris, France, September 27-29, 2006, LNCS 4195, pp. 159-171.

[8] A Loser, et al, Semantic overlay clusters within super-peer networks. Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2003), Berlin, Germany, September 7 - 8, 2003, LNCS 2944, pp. 33-47.

[9] A Montresor, A robust protocol for building superpeer overlay topologies. Proceedings of the Fourth International Conference on Peer-to-Peer Computing (P2P 2004), August 25-27, 2004, pp. 202-209.

[10] M Paolucci, T Kawmur, T Payne and K Sycara, Semantic matching of web services capabilities. Proceedings of the First International Semantic Web Conference (ISWC 2002), LNCS, 2342, pp. 333-347.

[11] PeerSim: a peer-to-peer simulator. http://peersim.sourceforge.net/

[12] A Rowstron and P Druschel, Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. Proceedings of the 18th IFIP/ACM Conference on Distributed Systems Platforms (Middleware 2001), Heidelberg (D), November 2001, LNCS 2218, pp. 329-350.

[13] I Stoica, et al, Chord: a scalable peer-to-peer lookup service for Internet applications. Proceedings of the 2001 ACM SIGCOMM Conference, pp. 149-160.

[14] R Studer, V R Benjamins and D Fensel, Knowledge engineering: principles and methods. Data and Knowledge Engineering, 25(1-2):161–197, 1998.

[15] Sun Microsystems, Jini architecture specification version 2.0, 2003. http://www.jini.org/

[16] P Triantafillou, C Xiruhaki, M Koubarais and N Ntarmos, Towards high performance peer-to-peer content and resource sharing systems. Proceedings of the Conference on Innovative Data Systems Research (CIDR 2003).

[17] UPnP Forum, UPnP device architecture version 1.0, 2000. http://www.upnp.org/

[18] M Vazirgiannis, K Nørvåg and C Doulkeridis, Peer-to-peer clustering for semantic overlay network generation. Proceedings of the 6th International Workshop on Pattern Recognition in Information Systems (PRIS 2006), Paphos, Cyprus, May 2006.

[19] B Yang and H Garcia-Molina, Designing a super-peer network. Proceedings of the IEEE International Conference on Data Engineering (ICDE, 2003), pp. 49-60.

PAPER C: Efficient Service Discovery System Based on Semantic Overlay Networks

Shanshan Jiang and Finn Arve Aagesen

Published in
Proceedings of the 6th International Information and Telecommunication Technologies Symposium (I2TS'07)

Brasilia, DF, Brazil, December 12-14, 2007.

Efficient Service Discovery System Based on Semantic Overlay Networks

Shanshan Jiang

Finn Arve Aagesen

Abstract A network of autonomous directories forming a large-scale distributed service discovery system is considered. In order to locate services efficiently in such large-scale systems, we propose to organize directories into Semantic Overlay Networks (SON) based on service ontology. The various SONs are further organized in super-peer networking structures. The focus is on the functionality for organization of directories into SONs and the use of SONs for efficient service discovery and efficient SON management. Efficient service discovery means high recall, small number of messages-per-request and small number of hops-per-request. Efficient SON management is characterized by small management procedure overhead, small load factor and short self-organization time. Simulations indicate that, compared to a system based on a random overlay network, such super-peer based SON system can achieve high recall with small hops-per-request value and significantly reduced messages-per-request value. Moreover, such system requires only a small management procedure overhead and the self-organization time is short both for SONs initial construction and reconstruction under dynamic node joining and leaving situations. Simulations also indicate a small average load factor.

1. Introduction

In a distributed service environment, service discovery is a core functionality to locate desired services. *Service discovery* is the process of finding the desired services by matching *service descriptions* against *service requests*. A *service description* provides service-related information which can be advertised by a service provider and searched during service discovery process. A *service request* represents user's service requirements. Both service descriptions and requests comprise information on functional and non-functional properties. Ontologies are the basis for adding semantic expressiveness to service descriptions and requests. An *ontology* is an explicit and formal specification of a shared conceptualization [1]. A *service ontology* is accordingly an explicit and formal specification of core concepts of the functional and non-functional properties of service. An *upper ontology of service* is a model of common service-related concepts applicable to a wide range of domains. *Semantic service discovery* is a service discovery process based on ontology concepts. Likewise, *semantic matching* is the matching of service requests and service descriptions based on ontology concepts.

Current service discovery protocols, such as SLP [2], Jini [3] and UPnP [4], are targeted for enterprise and home networks. They either rely on centralized entities for discovery or search entire local network by broadcast or multicast, thus do not suit for

wide area discovery. UDDI [5], the standard for Web service discovery, is based on centralized registries, also facing the scalability and performance challenges when the amount and variety of services increase. Hence, an important research objective is the location of services in a large-scale system with a huge amount of a large variety of services *efficiently*, *i.e. to answer service requests fast with low overhead*.

A large-scale service discovery system is considered, which consists of autonomous directories, where each directory has its own local registered service descriptions. We propose to apply semantic service discovery and to organize directories into *Semantic Overlay Networks* (SON). SON is a flexible network organization that logically connects nodes with semantically similar contents. The concept of SON is introduced in [6], which aims to improve query performance while maintaining a high degree of node autonomy. In this paper, a super-peer [7] based networking architecture is proposed for the various SONs. Super-peer architecture aims to combine the efficiency of a centralized approach and the scalability, load-balancing and robustness of a distributed approach. A super-peer architecture organizes nodes into hierarchy by utilizing the different capacities of nodes. Super-peers are nodes with high capacity. The super-peers and other nodes constitute the service discovery system. The super-peers have special assigned functionality in the service discovery procedure. They are dynamically selected by some super-peer selection algorithm and do accordingly not constitute single points of failure.

The focus of this paper is on the functionality for organization of directories into SONs and the use of SONs for efficient service discovery and efficient SON management. For the organization of directories into SONs, a shared service ontology based on a predefined service category hierarchy is used. The service descriptions are classified into service categories based on service ontology, and directories with semantically similar descriptions are grouped into SONs, where *semantically similar* means belonging to the same service category subtree. Each service request also identifies one service category. Service discovery based on SONs is an iterative process that can consist of several loops. Each loop has two steps. In Step 1, SONs which may contain relevant directories are selected. In Step 2, all the directories in these SONs are searched based on semantic matching. In each loop, the results are checked to see if enough number of matches is found. If not, more SONs are selected in new loops and until enough number of matches is obtained. For SON management, a self-organizing process based on gossiping [8] is applied for the construction and maintenance of SONs.

The rest of the paper is organized as follows: Sect. 2 discusses the system requirements, while Sect. 3 presents a model for such SON-based service discovery system. Sect. 4 describes the super-peer based SON system. Experiments and results are described in Sect. 5. Related work is discussed in Sect. 6 followed by conclusions in Sect. 7.

2. Requirements to an Efficient SON-based Service Discovery System

An efficient SON-based service discovery system must both be efficient with respect to *service discovery* and with respect to *SON management*. Efficient service discovery requires that relevant quality answers must be returned fast with small discovery

procedure overhead. Efficient SON management means fast construction and reconstruction of SONs and small management procedure overhead.

With respect to the efficient service discovery requirement two design requirements can be defined.

- **R1:** *The system should enable high probability searching with high recall*.

The number of SONs selected for answering a request should be small and at the same time contain directories that have a high number of matches. This is because that if the directories to which the request is sent have many semantically similar service descriptions (i.e. many possible matches), the request is answered fast.

- **R2:** *The number of SONs and the size of SONs must be small.*

Smaller SONs and fewer SONs reduce management procedure overhead. At the same time, fewer messages are required for answering a request, so the request is answered faster. Reducing SON connections per directory can reduce SON size. Shared service ontology with a predefined service category hierarchy is assumed. It is however inefficient to build SONs for each category. Aggregation of services into groups is needed, i.e. service categories at appropriate hierarchy level should be selected according to the service distribution so that the number of SONs and the SON connections can be reduced.

3. SON-based Service Discovery System Model

A *service discovery system* Ψ can be modeled as $\Psi = <O, D, L, F>$, where O is the *service ontology*, D is a set of directories, L is a set of *links* that logically connects directories based on semantic similarity and F is a set of *functionality*.

The service ontology O defines the service categories as well as other service related concepts in a hierarchical structure. The upper ontology of service *s* is defined as a tuple <*c, p, op, qos, sp*>, where *c* denotes the *service category* the service *s* belongs to, *p* the *policies* applied to *s*, *op* the *operations s* performs, *qos* the *QoS parameters* and *sp* the *service parameters*. As shown in Fig. 1, a *service* belongs to a *service category*, and has *operations, policies*, *service parameters* and *QoS parameters*. Each operation is defined by *inputs*, *outputs*, *preconditions* and *effects*. The set of categories $\{c\}$ is connected by a rooted tree, the *category hierarchy* $\mathcal{CH}$. The role and the use of the service categories are explained at the end of this Section. More details of definitions of service ontology elements and their application in semantic service discovery can be found in our previous work [9].

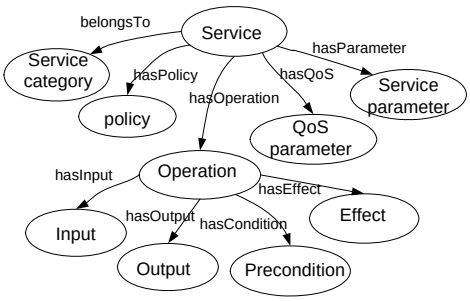


Fig. 1. Upper ontology of service.

$\mathbb{D}$ is the set of directories $\{d_i\}$. Each d_i maintains a set of *service descriptions* $SD_i = \{sd_j\}$. Each service description sd_j belongs to one service category c, denoted as $sd_j \rightarrow c$. Each d_i is *logically* connected with a set of directories, called its *neighbors* N_i.

$\mathbb{L}$ is a set of *links* $\{l\}$ that logically connects directories based on service categories. A *link* l is a triple $<d_i, d_j, c>$ where d_i and d_j are the logically connected directories based on a service category c. A link is symmetric, i.e., $<d_i, d_j, c>$ is the same as $<d_j, d_i, c>$. A *Semantic Overlay Network (SON)* is represented by the set of links with the same c, $SON_c = \{d_i, d_j \in \mathbb{D} \mid \exists \text{ a link } l = <d_i, d_j, c>\}$. Each directory can join one or several SONs.

$\mathbb{F}$ is a set of *functionality* provided by the system, which includes seven operations as described as follows. Each SON_c supports three operations:

- $Join_c(d_i, c)$: a directory d_i is added to SON_c, which means to find a directory $d_j \in SON_c$ so that a link $<d_i, d_j, c>$ can be added to the system Ψ.
- $Search_c(r, c)$: the service request r is sent to all the directories in SON_c and a set of matches are returned.
- $Leave_c(d_i, c)$: a directory d_i leaves SON_c by simply dropping all the links d_i maintains.

In addition to the above three operations related to SONs, the service discovery system also supports the following operations:

- *Register*(sd_j, d_i, c): a service description sd_j is registered in a directory d_i with service category c.
- *Assign*(d_i): a directory d_i is assigned to a number of SONs according to the service descriptions registered. In addition, SON memberships can be updated according to runtime situation.
- *Discovery*(r, c): a service request r with a service category c specified by the user is sent to the system for discovery and the results are sent back to the user.
- *SONselection*(c): select the relevant SONs for a given service category c.

The role and the use of the service categories can be explained as follows. In service ontology, the category hierarchy $\mathcal{CH}$ is defined and is referenced by both users and service providers when they define service requests and descriptions. Each service description is mapped to one service category when it joins the system. Each service request contains one service category c_r which defines the range of services that the user is looking for. Each SON_c is related to one service category c, which is the root of the category subtree from $\mathcal{CH}$. Fig. 2 illustrates the category relations in a directory. The black circles represent categories related to service descriptions, while larger circles represent categories related to SONs. Each directory has a set of service categories from all the service descriptions registered in the directory (the set *ServiceDescriptions*). By aggregation of service descriptions according to their semantic relationships, a new set of categories (the set *Potential_SONs*) is obtained. This category set corresponds to potential SONs. Further applying join policies, a new set of categories of $\{c'\}$ (the set *SONs*) is obtained (cf. Sect. 4.1). Each c' corresponds to one $SON_{c'}$ that the directory is member of. A directory joining $SON_{c'}$ implies that it has enough service descriptions that fall under the subtree c'.

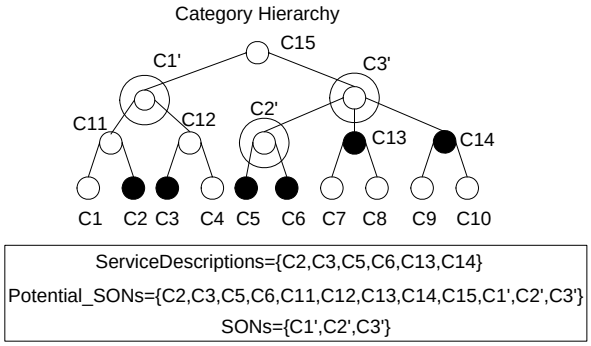


Fig. 2. Category set examples as seen from one directory.

4. A Super-peer Based SON Service Discovery System

The architecture for a super-peer based SON service discovery system is illustrated in Fig. 3. Service descriptions are stored in directories, while service requests from a service discovery user are sent to the service discovery system via the *edge* directory. Directories are logically organized into SONs. Each directory can belong to several SONs. The *capacity* of a directory is characterized by the number of directories it can manage. Each SON has a *super-peer*, which is a directory with high capacity. Other directories in a SON are all connected with the super-peer and communicate via the super-peer. In addition, such super-peers are connected with each other for inter-SON communication. In other words, the super-peer can distribute messages within a SON as well as forward them to different SONs for global discovery. The inter-SON communication is basically broadcast-based, although other approaches can be applied, such as DHT (Distributed Hash Table) [10].

Each directory in the service discovery system can have one or several of the following roles: *service storage*, *edge* and *super-peer*. Each directory has at least the role of *service storage*, which has the following functionality: 1) Registering service descriptions, 2) Assigning the directory to SONs and adapting SON memberships to dynamic changes, and 3) Accepting service requests from a super-peer and carrying out local semantic matching procedure. The directory that a service discovery user contacts also assumes the *edge* role, which will 4) accept the service request from the service discovery user, forward it to its super-peer (can be the same directory), and return the answers to the user. Furthermore, each SON has a *super-peer*, which has the following functionality: 5) selecting relevant SONs for discovery, forwarding requests to other directories in the same SON and/or other relevant SONs accordingly, and returning results back to the edge. A sequence diagram showing the interactions between different roles during service discovery will be given in Sect. 4.3.

In addition, system functionality is needed for 6) SON super-peer network construction and maintenance, including directory joining and leaving.

Each service is registered in a local directory with a service category. Policies are applied when assigning directories to SONs. Service discovery requires the iterative selection of SONs and searching in each SON. In the following, assignment of directories to SONs, SONs construction and maintenance as well as service discovery are explained in more detail.

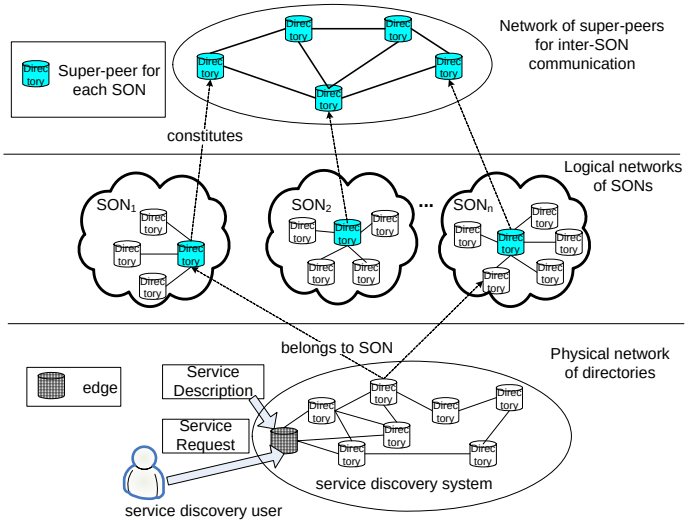


Fig. 3. Super-peer based SON service discovery system architecture.

4.1 Assignment of Directories to SONs

To join the SON-based service discovery system, a directory needs to find out which SONs to join based on the service categories of the service descriptions contained in it. To meet requirement R2, a directory should first aggregate service categories of all service descriptions stored in it into groups, and then select the best "representative" service categories to join SONs according to join policies. As services can dynamically join and leave the system, the services registered in the directories can change, causing SON membership to change. The join policies will thus not only be applied at startup, but also be applied dynamically for directories to adapt to runtime situation.

4.1.1 Aggregation

Aggregation is based on the semantic relationships between service categories. For two service categories c and c', $c' \preceq c$ means c' equals to c or c' is a descendant of c in the service category hierarchy. To aggregate service descriptions, a service description sd will be placed in a group of service category c (denoted as $sd \Rightarrow c$) if ($sd \rightarrow c'$) and ($c' \preceq c$). In other words, c is the root of subtree that c' belongs to.

4.1.2 Join Policies

Assignment of directories to SONs is based on the group information which is the result of the aggregation. In the following three join policies Join1-Join3 are defined. The selection of join policies is determined by the distribution of services. If services are clustered, each directory contains only a few service category groups. Both the number and size of SONs can be small if applying Policy Join1. On the other hand, if services are evenly distributed, Policy Join2 and Join3 need to be applied to reduce the SON connections.

Policy Join1: A directory d_i joins a SON_c if it has a service description that belongs to group c, i.e. ($d_i \in SON_c$) if ($\exists\ sd_j \Rightarrow c$, $sd_j \in d_i$). This is the most conservative policy, but it tends to produce many SON connections.

Policy Join2: A directory d_i joins a SON_c if the number of service descriptions that belongs to *c* is greater than the *join threshold* T, i.e. ($d_i \in SON_c$) if (Count($sd_j \Rightarrow c$) > T). Such policy is good and practical for wide-area service discovery. If a directory d_i joins a SON_c, this *implies* that d_i contains many service descriptions falling under the subtree *c*; therefore, selecting d_i for matching will provide higher number of matches than selecting the directories that are not members of SON_c. Policy Join2 thus meets the *high probability searching* requirement (R1). The join threshold T is an adjustable parameter. Policy Join1 can be viewed as setting T=0. The value of T is important for the performance of SON-based service discovery system. Taking a high T value, SON connections can be reduced, and accordingly SON sizes are reduced (R2). However, for small groups of service descriptions that are not assigned to SONs, there will be possibility that such descriptions never be searched if they are not associated with any SON. In this case, Policy Join3 can be applied.

Policy Join3: For service groups that do not have enough service descriptions, the directory will join a SON that corresponds to an ancestor of current service category.

4.2 SONs Construction and Maintenance

To meet requirement for efficient SON management, a self-organizing process is applied for both SONs construction and maintenance based on gossiping [8]. It is an autonomous functionality determined by directories themselves and enables the updates of topology information. In particular, failed or leaving nodes are automatically deleted by the absence of information exchange. Initially, all the directories in the system decide which SONs to join (i.e. select appropriate service categories) according to the function described in Sect. 4.1. For each SON, one directory with high capacity will be selected as super-peer, while other directories are *clients* of the super-peer. Each super-peer has a maximum capacity which determines the maximum number of directories it can manage. If a large number of directories decide to join SON_c, several super-peers may be needed. This means that there are several SONs corresponding to the same category *c* and each of them is managed by a super-peer.

A modified version of SG-1 algorithm [11] is adopted for super-peer selection. This algorithm is based on gossiping and assures the dynamic selection of super-peers according to runtime situation. Each node periodically exchanges its current status information (called *partial view*) with randomly selected peer nodes, which contains information such as identifier, capacity, neighbors, SON memberships, current role in a SON (super-peer or client), and the number of clients they are serving. After exchanging views, nodes with available connections and higher capacity will be changed into super-peers, and others are to connect to them as clients. Alternatively, a super-peer may decide to move all its clients to another super-peer with more capacity, and become a client itself. Such process continues until the system becomes stable, i.e., the minimum number of super-peers is selected. The super-peer selection process is also applied when directories join or leave the system (i.e. SON maintenance). In addition, each client periodically probes its super-peer to see if it is active. If the super-peer fails or leaves, the client can declare itself as a super-peer, and participate in the above super-peer selection process. For more details, please refer to our previous paper [12].

4.3 Service Discovery

When a request is sent to the service discovery system, the discovery operation *Discovery*(r, *c*) is carried out by an iterative procedure consisting of two steps in each loop.

4.3.1 Step 1: Select Relevant SONs

This corresponds to the operation *SONselection*(*c*). Given a service request with category *c*, the system needs to map it to SONs which may contain relevant service descriptions. The selection of relevant SONs is determined by the semantic distance between *c* and c_s (category for a candidate SON) in the $\mathcal{CH}$. An *iterative selection policy* is adopted, which selects SONs associated with *c*, and both its descendants and ancestors *c*' in an order that can enable *high probability searching* (R1). SONs most likely to answer the request are selected first to carry out Step 2 (cf. the following subsection). Then less probable SONs are selected until enough number of matches is obtained. In detail, given a request for category *c*, SON_c (if it exists) is searched first. If not enough number of matches is obtained, SONs that correspond to *c*'s descendents (if any) are searched. If still not enough number of matches returned, SONs that correspond to *c*'s ancestors are searched (if any, and from the nearest ancestor until the root of the $\mathcal{CH}$). Such process continues until enough number of matches is obtained or every possible SONs are searched. Since the maximum size of each SON is determined by the capacity of its super-peer, there may be several SONs corresponding to one category *c*. In this case, each such SON is selected until enough results are obtained.

4.3.2 Step 2: Search Each Directory in SON_c

This corresponds to the operation $Search_c$(r, *c*). Since each SON is managed by a super-peer, the super-peer will forward the service request to each directory in the SON. Then semantic matching is carried out in each directory. This is based on other service concepts than service category in the service ontology, such as operations, service and QoS parameters. The semantic matching is based on semantic similarity between ontology concepts. The results of a semantic service discovery can be ranked according to functionality similarity based on degree of match, which can be distinguished as *Exact, Plugin, Subsume* and *Fail* [13]. A match between a service description *sd* and a service request *r* can then be modeled as a function *M(sd, r)* = {1, if *degree* ∈ (*Exact, Plugin, Subsume*) | 0, if *Fail*}. Our previous paper [9] proposed an approach for semantic matching which is based on service ontology defined in Fig. 1.

An UML sequence diagram for service discovery procedure is given in Fig. 4. The arrows with operation names represent operational messages, while arrows without names are the return messages, containing results for operational messages. The frame **loop** refers to a loop behavior, while frame **opt** means optional behavior. Fig. 4 shows that a service request is first sent from the service discovery user to the edge, then forwarded to super-peer1 (the super-peer of the edge, which can be the edge itself). The loop Discovery in super-peer1 consists of iterations of SON selection, search within the same SON as well as forwarding the request to corresponding SONs (super-peer2) for

search. At the end of each loop, the results are checked to see if enough number of matches is obtained.

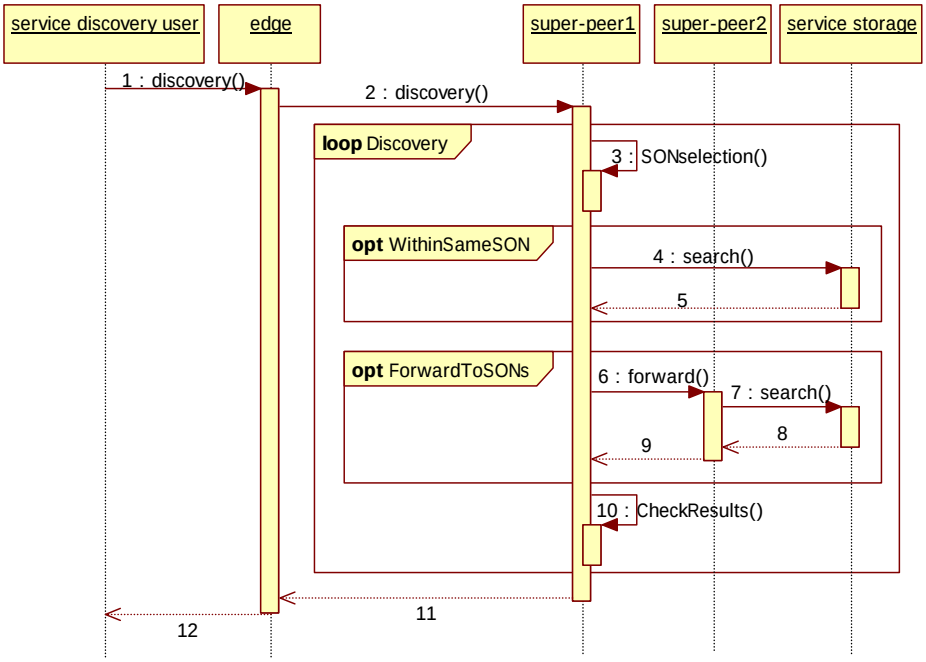


Fig. 4. UML sequence diagram for service discovery procedure.

5. Evaluation

5.1 Evaluation Measures

The service discovery system performance is characterized by: 1) *Service discovery efficiency* and 2) *SON management efficiency.* Measures for service discovery efficiency are: *recall, messages-per-request* and *hops-per-request*. Measures for SON management efficiency are *management procedure overhead, load factor* and *self-organization time.*

Recall of a discovery process is the proportion of relevant service descriptions that are retrieved out of all relevant service descriptions. Higher recall indicates better discovery quality. *Messages-per-request* is the number of messages generated by the discovery system to answer a service request. This represents discovery procedure overhead. The larger the messages-per-request, the more overhead and longer time it needs to answer a request. *Hops-per-request* is the average number of hops for answering a request. The smaller the hops-per-request, the faster the request can be answered.

Concerning SON management efficiency measures, the number of messages sent per directory per round in order to maintain super-peer based SON structures is applied as a measure for *management procedure overhead*. The *load factor* is defined as the value of *management procedure overhead* divided by directory *capacity*. A small load factor indicates a small work load for a directory to maintain the super-peer based SON structures. As a measure for *self-organization time*, the number of simulation rounds for the system to stabilize is used. A smaller number of rounds indicate that the system can stabilize faster.

5.2 Experiments

5.2.1 Service Discovery Efficiency

In these experiments, SONs are constructed first and then service discovery performance is evaluated in terms of recall, messages-per-request and hops-per-request. We have written a simulator in Java and used BRITE [14] to generate network topologies. Assume the classification of service descriptions and requests is correct and accurate, so that only one SON_c is selected for *Discovery*(r, c_r). The network size varies from 1000 to 10000 nodes (directories), with 2500000 service descriptions falling into 250 service categories. 25000 service requests are randomly generated for each network size. Power-law distribution on node capacities is applied. The maximum capacity is 2000. Since the number and size of SONs are influenced by service distribution, two cases are simulated: 1) *uniform*, where service categories as well as service descriptions are uniformly distributed to directories. This represents the case when services are *evenly distributed*; 2) *power-law*, where the distribution of service categories on directories follows power-law and service descriptions are uniformly distributed to categories in the directories. This represents the case when services are *clustered*. For each case, 10 replications with different seeds have been conducted for each network size. The results of our SON-based system are compared with Gnutella-based [15] pure P2P system, where nodes are connected by a random overlay network and flooding is used for routing requests. Gnutella system is not sensitive to service distribution. Its recall and messages-per-request are determined by TTL (time-to-live) parameter.

Fig. 5 shows the average number of messages-per-request for achieving recall of 1.0 for different network sizes. It shows that, to retrieve all relevant service descriptions, the average messages-per-request for Gnutella system is about 3 to 4 times of that needed for SON uniform system (i.e. SON system with uniform distribution), and about 7 to 46 times of that needed for SON power-law system (i.e. SON system with power-law distribution). SON uniform system needs about 3 to 10 times of the messages-per-request required for SON power-law system. As the network size increases, the overhead of Gnutella system in terms of messages-per-request increases much more than that of SON system. Assume the values of messages-per-request are normally distributed, the confidence intervals (CI) can be determined. Take an example, for SON uniform system with 1000 nodes, the average messages-per-request is 861.7, its standard deviation is 7.16, and the 95% confidence interval will be $856.58 \leq CI \leq 866.82$.

Fig. 6 shows the average messages-per-request for different recall for network size of 1000 and 2000. It shows that, to achieve the same recall (by adjusting TTL for Gnutella system and join threshold T for SON uniform system respectively), the average messages-per-request sent by Gnutella system is about 3 to 4 times of that needed for SON uniform system.

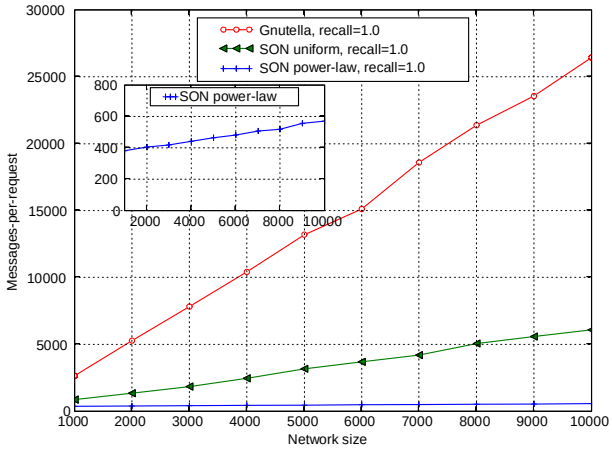


Fig. 5. The average number of messages-per-request for recall=1.0.

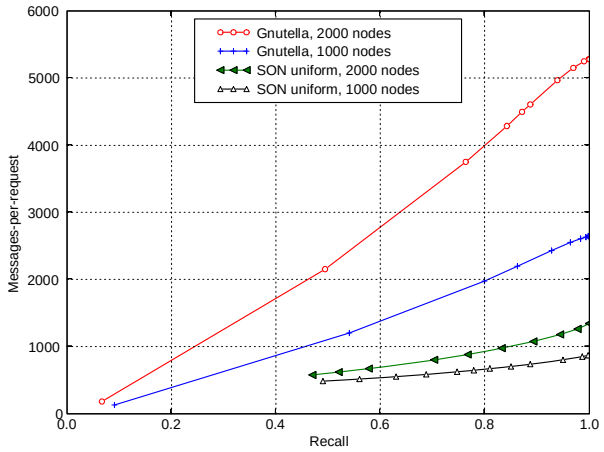


Fig. 6. The average number of messages-per-request for different recall for network size 1000 and 2000.

The number of hops for answering a request is determined by the hops to find the super-peer for SON_c, i.e. by the inter-SON communication. If flooding is used for inter-SON communication, the results from simulation give a hops-per-request value between 4 and 5 for different network sizes. This indicates a service request in SON system can be answered fast.

The size and number of SONs determine the messages-per-request for discovery. If the SON connections per node are reduced, the SON sizes as well as the messages-per-request will be reduced. When services are clustered, each node will have a small number of service groups and will have a few SON connections. Table 1 shows the effect of service distribution and join policies on system performance for network size of 2000. The applied policy will be Join1 when the adjustable join threshold parameter T=0 and Join2 with T>0 (cf. Sect. 4.1.2). When the threshold parameter T=7, a node will only join SON_c when it has more than 7 service descriptions belonging to group *c*. The SON connections per node and SON size will accordingly be changed by the variation of T. As a consequence, the service discovery efficiency measures will also change. Gnutella system is also listed for comparison and its recall and messages-per-

request will be changed when adjusting TTL. As can be seen from the table, when applying Policy Join1 (i.e. T=0), the average SON size for SON uniform system is about half of the network size, while the average SON size for SON power-law system is about one-fiftieth of the network size. SON power-law system has smaller number of SON connections per node, and the messages-per-request is only about one-thirteenth that of Gnutella system (for same recall). SON uniform system has higher number of SON connections per node, and the messages-per-request is only about one-fourth of Gnutella system (for same recall). By applying Policy Join2 and adjusting T, SON connections are reduced, thus SON size and messages-per-request are reduced accordingly, as shown in Table 1 (SON Uniform, T=7). The recall is also reduced in this case; however, SON Uniform system needs only one-fourth of the messages-per-request value required by Gnutella system to achieve the same recall. The recall may be increased by iterative selection of SONs for discovery (e.g. SONs corresponding to an ancestor of category *c*), but the messages-per-request will also be increased.

TABLE 1. SOME SYSTEM PERFORMANCE MEASURES FOR NETWORK SIZE=2000.

SON system						
Distribution	T[1]	*SONs/node*[2]	*SON size*	*Recall*	*Msg*[3]	*Hops*[4]
Uniform	0	123	985	1.0	1349	4
Uniform	7	65	521	0.77	884	4
Power-law	0	5	42	1.0	405	4
Gnutella system						
TTL[5]				*Recall*	*Msg*[3]	
181				1.0	5281	
4				0.76	3751	

Note: 1. Join threshold parameter. 2. Average number of SON connections per node. 3. Messages-per-request. 4. Hops-per-request. 5. "Time-to-live" parameter.

The results indicate that when services are clustered, SON system can improve discovery efficiency with high recall and significantly reduced messages-per-request. Even when services are evenly distributed, SON system still has better discovery performance than Gnutella system, requiring much smaller messages-per-request. Moreover, policies can be applied to ensure a few SON connections per node and keep SON size small, so that messages-per-request can be reduced. There is however a tradeoff between high recall and small messages-per-request when services are evenly distributed.

5.2.2 SON Management Efficiency

For these experiments, the PeerSim simulation framework [16] is used. To realize the $Search_c(r, c)$ and $Join_c(d_i, c)$ operations, a general super-peer based protocol for communication within individual SON is implemented and an existing implementation of Newscast – a gossip-based membership protocol [16] is used for inter-SON communication. Assume each directory joins one SON. SONs are constructed according to procedure defined in Sect. 4.2. Power-law distribution on directory capacities is applied. Simulation rounds for network sizes from 1000 to 50000 are carried out. To simulate dynamic behavior of the system, 100 nodes are deleted while 50 new nodes are added to the system in each round from round 30 to round 35.

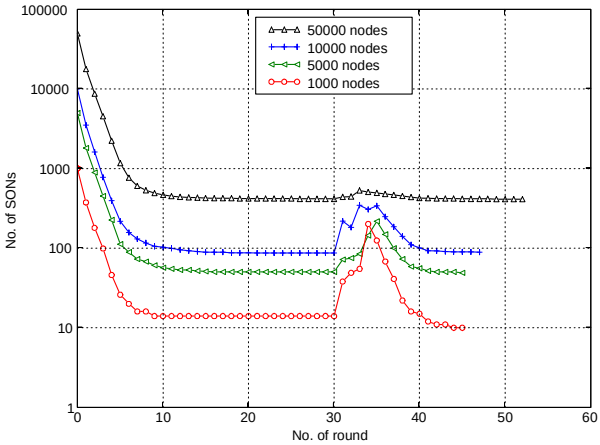


Fig. 7. Self-organization time (number of rounds) for different network sizes.

Fig. 7 shows that our proposed SON-based system becomes stable in approximately 10 rounds in both SONs initial construction and reconstruction under dynamic situation. Fig. 8 shows that average management procedure overhead is between 1 and 6. Fig. 9 shows that maximum management procedure overhead is usually below 15 for network size≤10000, and below 20 for network size=50000. Fig. 10 shows the maximum and average load factor for each round for network sizes of 1000 and 5000. The average load factor is below 1 and maximum load factor below 13. It also demonstrates strong correlation with Fig. 7 and indicates a very small value of load factor when the system is stable and a relative high value during the self-organizing process for initial SONs construction and reconstruction under dynamic situations. This can be explained as follows. Most of the management procedure overhead is related to the super-peer selection process. Initially, nodes with low capacity also participate in this process. This leads to a high load factor value. After one message exchange, such nodes will be changed into clients, and leave the selection process. Gradually, only nodes with higher capacity (potential super-peers) will participate in the selection process, reducing load factor (cf. Sect. 4.2).

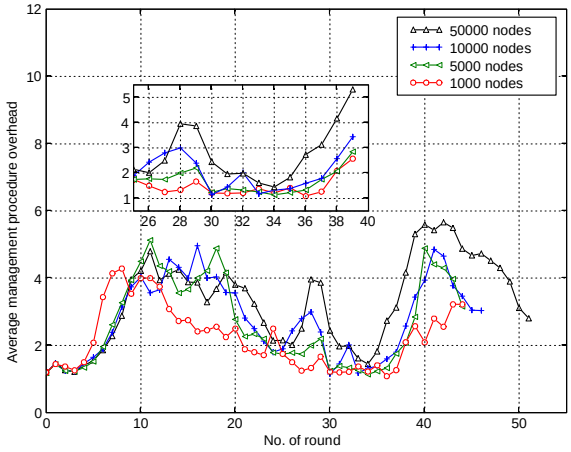


Fig. 8. Average management procedure overhead for different network sizes.

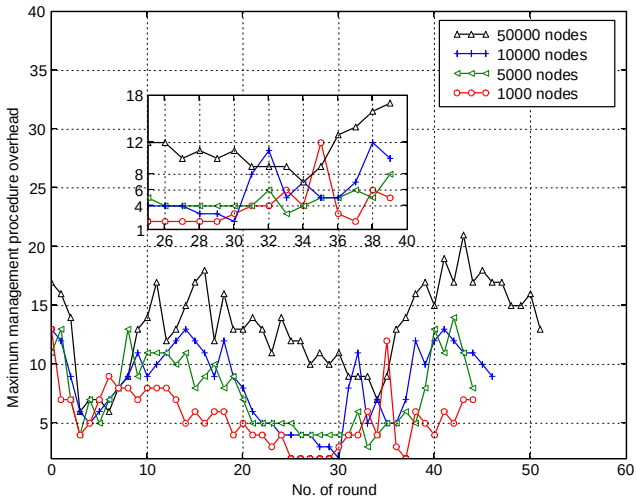


Fig. 9. Maximum management procedure overhead for different network sizes.

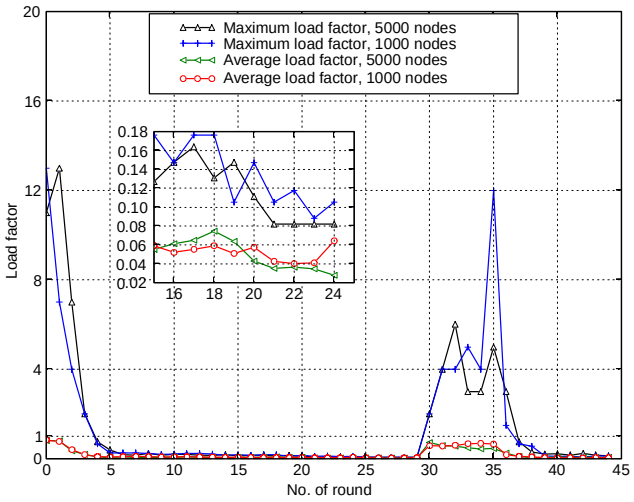


Fig. 10. Maximum and average load factor for network size of 1000 and 5000.

6. Related Work

P2P-based service discovery systems have been proposed as promising solutions providing scalability and autonomy. Pure P2P-based service discovery systems [17] based on flooding for routing requests, do not rely on centralized entities and are well-suited for highly dynamic environments such as ad hoc networks. However, flooding increases message overhead and is not suitable for wide area discovery.

DHT-based P2P systems [10][18] can locate relevant nodes with a bounded number of messages by applying the DHT algorithm. However, as DHT is based on a single key value, DHT-based system can not process complex queries in service discovery (e.g. based on ontology). Moreover, DHT techniques either require a specific network structure or assume total control over the location of the data. The high maintenance

cost for DHT-based index and the lack of node autonomy limits the scalability and the application to wide area service discovery systems.

Lately several approaches have been proposed for creating and using clusters or SONs to improve search in P2P systems. SONs based on pure P2P systems (i.e. random overlay networks) have been proposed in [19][20]. In [21], clustering policies are proposed to generate semantic clusters in super-peer networks. Our approach differs by clustering directories based on service category hierarchy. Furthermore, our approach considers the adaptation of SONs, where the number of SON connections and SON membership can change dynamically by applying join policies on each directory at runtime. In addition, the super-peers in our system do not index data of their clients and are mainly used to route requests to proper SONs.

This paper adopts a modified version of SG-1 [11] for super-peer selection. A recent protocol and a natural evolution of the SG-1 algorithm is SG-2 [22], which introduces the notion of latency between peers and poses a QoS limit on it. In addition, SG-2 is strongly bio-inspired.

7. Conclusions

Semantic Overlay Networks (SONs) have been proposed as the basis for improving the discovery efficiency in large-scale service discovery systems. The focus of the paper is on the functionality of the organization of directories into SONs and the use of SONs to locate services efficiently as well as efficient SON management. A super-peer based SON service discovery system is described. Aggregation and join policies are applied to ensure the SON size and the number of SONs can be small. The number of SON connections and SON membership can change dynamically by applying join policies at runtime. Iterative selection of SONs for discovery enables high probability searching. A self-organizing construction and maintenance of SONs ensures efficient SON management. The proposed system has been evaluated by simulations. Simulations indicate that such super-peer based SON system can significantly improve discovery efficiency by providing high recall with small messages-per-request and small hops-per-request. The simulations also indicate a small management procedure overhead and short self-organzation time. It also indicates a small average load factor. The results, however, indicate that a SON system has better performance when services are clustered than when services are evenly distributed.

References

[1] Studer, R. et al. (1998). Knowledge engineering: principles and methods. *Data and Knowledge Engineering, 25(1-2)*:161–197, 1998.

[2] Guttman, E. et al. (1999). Service location protocol, version 2. RFC2608.

[3] Sun Microsystems. (2003). Jini architecture specification version 2.0. http://www.jini.org/

[4] UPnP Forum. (2000). UPnP device architecture version 1.0. http://www.upnp.org/

[5] UDDI. (2002). Universal description, discovery and integration of web services. version 1.0. http://www.uddi.org/

[6] Crespo, A., & Garcia-Molina, H. (2002). Semantic overlay networks for P2P systems. *Technical Report*, Computer Science Department, Stanford University, October 2002.
[7] Yang, B., & Garcia-Molina, H. (2003). Designing a super-peer network. In *Proc. of ICDE'03*.
[8] Demers, A. et al. (1987). Epidemic algorithms for replicated database maintenance. In *Proc. of POCD'87*.
[9] Jiang, S., & Aagesen, F.A. (2006). An approach to integrated semantic service discovery. In *Proc. of AN'06*.
[10] Stoica, I. et al. (2001). Chord: a scalable peer-to-peer lookup service for Internet applications. In *ACM SIGCOMM'01*.
[11] Montresor, A. (2004). A robust protocol for building superpeer overlay topologies. In *Proc. of P2P'04*.
[12] Jiang, S. et al. (2007). A self-organizing service discovery system based on semantic overlay networks. In *Proc. of CODS'07*.
[13] Paolucci, M. et al. (2002). Semantic matching of web services capabilities. In *Proc. of ISWC'02*.
[14] Medina, A. et al. (2001). BRITE: an approach to universal topology generation. In *Proc. of MASCOTS'01*.
[15] Gnutella. (2003). The Gnutella protocol specification v0.4.
[16] PeerSim. (2003). PeerSim: a peer-to-peer simulator. http://peersim.sourceforge.net/
[17] Paolucci, M. et al. (2003). Using daml-s for p2p discovery. In *Proc. of ICWS'03*.
[18] Rowstron, A., & Druschel, P. (2001). Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware*, *2001*.
[19] Triantafillou, P. et al. (2003). Towards high performance peer-to-peer content and resource sharing systems. In *Proc. of CIDR'03*.
[20] Vazirgiannis, M. et al. (2006). Peer-to-peer clustering for semantic overlay network generation. In *Proc. of PRIS'06.*
[21] Loser, A. et al. (2003). Semantic overlay clusters within super-peer networks. In *Proc. of DBISP2P'03*.
[22] G. P. Jesi, A. Montresor and O. Babaoglu, Proximity-aware superpeer overlay topologies. In *Proc. of SelfMan'06.*

PAPER D: XML-based Dynamic Service Behaviour Representation

Shanshan Jiang and Finn Arve Aagesen

Published in
Proceedings of Norsk informatikkonferanse (NIK'03)

Oslo, Norway, November 24-26, 2003.

XML-based Dynamic Service Behaviour Representation

Shanshan Jiang

Finn Arve Aagesen

Abstract This paper presents an XML-based framework for implementation language and platform independent service execution behaviour representation. It is based on an FSM-interpreter which can interpret and execute XML-represented FSM behaviour. This model has been integrated into TAPAS (*Telematics Architecture for Plug-and-Play Systems*) platform. TAPAS has basically two engineering viewpoint functionality classes: i) Dynamic service configuration and ii) Dynamic service instantiation. The dynamic service configuration is already based on XML. Using XML for the service execution representation gives one integrated representation of all dynamic service related functionality. As XML basically is a structure representation language, dynamic behaviour representation is made possible by utilising the inherent characteristics of the TAPAS basic architecture. TAPAS defines the behaviour by plug- and- play manuscripts and Node inherent capabilities.

Keywords: XML, EFSM, Plug-and-Play, service behaviour description

1. Introduction

Due to the increase in both heterogeneity and complexity in today's networking, wide-area distributed computing and service provision systems, there arises a demand for an architecture for network-based service systems that gives flexibility and efficiency in the definition, deployment and execution of the services and at the same time, takes care of the adaptability and evolution of such services.

The *TAPAS* project (TAPAS = *Telematics Architecture for Plug-and-Play Systems*) [1,2,3,4] is a research project, which aims at developing an architecture for network-based service systems with A) flexibility and adaptability, B) robustness and survivability, and C) QoS awareness and resource control. The goal is to enhance the flexibility, efficiency and simplicity of system installation, deployment, operation, management and maintenance by enabling dynamic configuration of network components and network-based service functionality.

Another objective is to gain experiences and knowledge by implementing those various features, both for demonstrating the implementation possibility and for validating the feature applicability. The goal is not to develop a complete executing architecture, but is to set the various features coming from the above defined requirements in a context related to totality. The TAPAS architecture consists of two service *functionality classes*: P) the basic architecture and Q) the mobility handling architecture, where Q) is an extension of P). Considering the functionality from an engineering viewpoint, the *engineering functionality* can be classified as i) Dynamic service configuration and ii) Dynamic service instantiation. *Dynamic service configuration* makes decision about in which Node to run the service software. This

decision is based on information about required and offered status and capabilities. *Dynamic service instantiation* comprises service creation, definition, deployment, execution and management.

The TAPAS architecture requires a support system for the engineering functionality. The support system for dynamic service instantiation is denoted as the TAPAS platform. The Dynamic service configuration functionality is based on the TAPAS platform, and the engineering dynamic configuration functionality can also be considered as a service functionality class R) in addition to the classes P)-Q) described above.

Parts of the specified support functionality have previously been implemented using Java RMI and Web technologies as a means for service definition, update and discovery. For more information of the architecture, implemented system and ongoing research activities, the reader is referred to [1,2,3,4] and the Web site http://tapas.item.ntnu.no/.

The TAPAS basic architecture is based on actors that can download manuscripts defining roles to be played. The ability to play roles depends on the defined required capability and the matching offered capability in a node where an actor is going to play. XML (eXtensible Markup Language) [6] is a standard for interchanging structured documents over the Internet. It has potential in offering interoperability across heterogeneous systems, and can be used to integrate systems by interchanging data and metadata (data about data). The TAPAS dynamic configuration functionality uses standard XML-based metadata and ontology languages for modelling and providing semantic description of capabilities and status of Plug-and-Play (PaP) systems [2]. By also using XML as a basis for the service instantiation functionality, we will have one common representation language through the whole architecture. This is the motivation for the work described in this paper. A model is presented, which uses XML as a representation language for the basic behaviour of the Extended Finite State Machines (EFSM), but where the actions are based on Node inherent capabilities. This model has been integrated into the TAPAS platform. The model is at the moment basically supporting the basic architecture P), and this paper is also concerned about this basic architecture. Parts of the mobility handling architecture Q) is already XML-based [5], but the solution presented here will be used as the basis for actor movement, which is a needed function for the fully support of all aspects of mobility handling [5].

The paper is organized as the following. Section 2 describes relevant parts from the TAPAS architecture. Section 3 gives the model and framework for XML-based behaviour modelling. Section 4 describes the implementation using Java technology and its integration into TAPAS platform. Conclusions are given in section 5.

2. TAPAS Basic Architecture and Dynamic Configuration Functionality

TAPAS has already been classified related to service and engineering functionality. This Section is focusing on TAPAS basic architecture and the dynamic configuration functionality.

The TAPAS basic architecture (Figure 1) is based on generic actors in the nodes of the network that can download manuscripts defining roles to be played. This model is founded on a theatre metaphor, where actors perform roles according to predefined

manuscripts, and a director manages their performance. Actors are software components, which represent functionality to be executed at different nodes within the network. Roles are modeled as EFSM. A director is an actor with supervisory status regarding all other actors' plug-in and plug-out phases. A director also represents a domain, which is a set of nodes managed by a single director.

A *service system* consists of *service components* which are units related to some well-defined functionality defined by a *play*. A play consists of several *actors* playing different *roles*, each possibly having different requirements on *capabilities* and *status* of the executing system. A role-session is a projection of the behaviour of an actor with respect to one of its interacting actors. An actor will constitute a *role figure* by behaving according to a *manuscript* defining the *functional behaviour* of a particular role in a play. A service component is realised by a *role figure* based on a *role* defined by a *manuscript*. A role figure, however, is realised in an executing environment in a node and is utilising capabilities. A *capability* is an inherent property of a *node* (or a *capability component*). A capability component may have several capabilities. These capabilities are offered to actors. The ability to play roles depends on the *defined required capability* and the *matching offered capability* in a node where an actor is going to play. Examples of capabilities are *processing, storage and communication resources* (e.g., CPU, hard disk and transmission channels), *standard equipment* (e.g., printers and media handling devices), *special equipment* (e.g., encrypting devices), and *data* (e.g., user login and access rights). Capability can be specialized into three subclasses: Function, Resource and Data [1]. Functions are pure software or combined software/hardware components used for performing particular tasks.

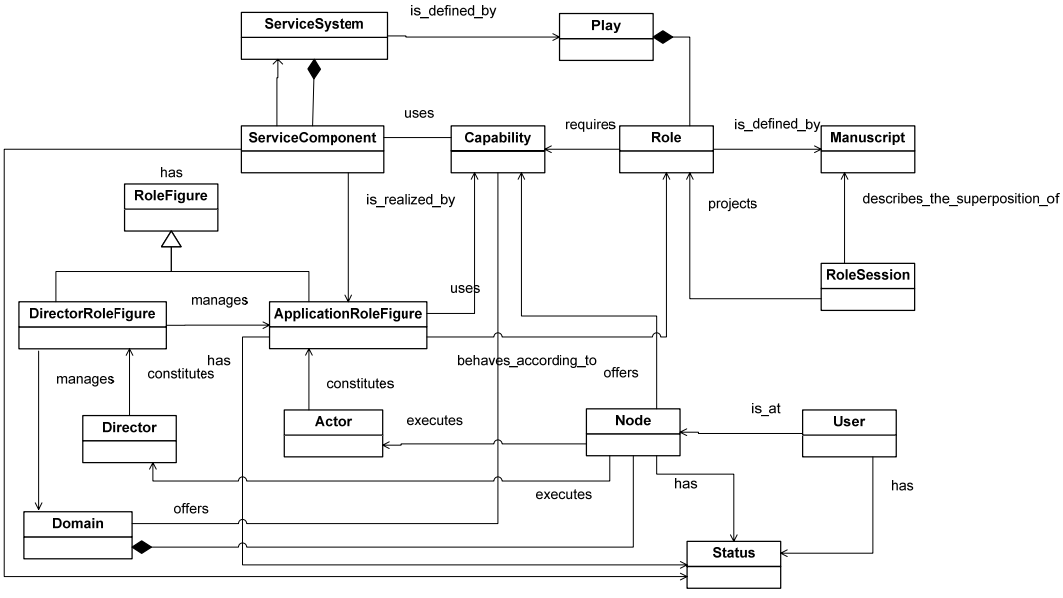


Figure 1. TAPAS basic architecture (object model)

Due to dynamic availability of nodes in the network as well as changes in their capabilities and status, configuration and reconfiguration of nodes to constitute particular service components must be computed on the fly. The TAPAS dynamic configuration functionality (Figure 2) is extended to deal with such a requirement [2].

As seen from Figure 2, the Play repository stores a collection of play definitions, each of which defines requirements and functional behaviours of a corresponding PaP service system. A play definition consists of four specifications: 1) Play configuration rules, 2) Reconfiguration rules, 3) Role specifications and 4) Manuscripts. Role specifications identify the requirements on available capabilities and status for each role. Role specifications and manuscripts define two aspects of roles in a play: the static and dynamic models. The former describes the metadata of each role and the latter models its functional behaviour in terms of EFSM. Hence, they provide information of "what the role is" and "how it is realised", respectively. Role specifications, play configuration and reconfiguration rules are uniformly formalised in a single representation schema, i.e., XML Declarative Description (XDD) [14].

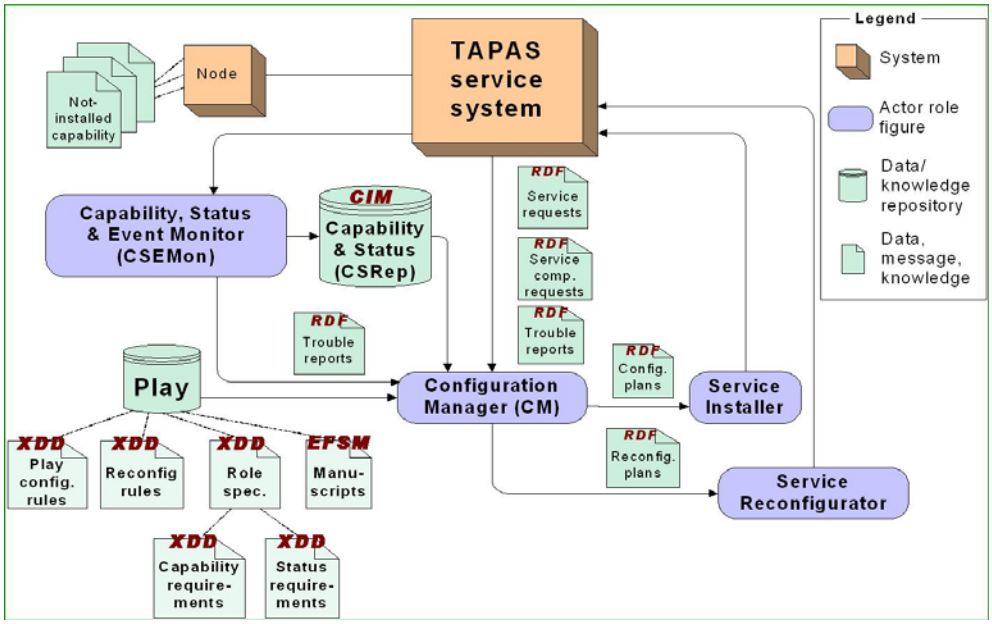


Figure 2. TAPAS dynamic configuration functionality

3. Behaviour Description Using XML

XML is appropriate for the modelling of structure and data. However, to use XML to represent service logic, i.e., the dynamic behaviour, poses a challenge. Some XML-based programming language are available, such as Superx++[8] and XL[9]. But the limitation for using such a language is that the language is not general enough and not portable.

In TAPAS, the behaviour is defined by plug-and-play manuscripts and Node inherent capabilities. The manuscript definition is basically a state machine specification of certain functionality, and describes the behaviour of the actor performing it. When a manuscript is executed, the internal actions of the state machine can utilise the functions provided by nodes, which are inherent capabilities of the nodes. On the other hand, once a manuscript is downloaded and a role is plugged-in on a node, the actions this role figure can perform represent new capabilities the node provides.

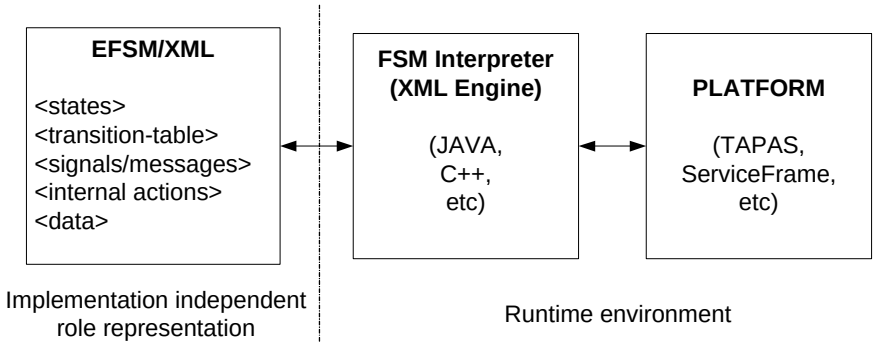


Figure 3. XML-based service system model

The model for representing EFSM using XML is illustrated in Figure 3. FSM interpreter is the core part for this model. It is the interface between XML-based EFSM description and the execution platform. Figure 4 gives the functionality for the FSM interpreter. The FSM interpreter has some basic functionality, which is same for all applications. This includes loading XML file into memory when initializing an FSM, signal/message processing (sending / receiving), internal actions, state transitions and FSM management. In addition, this FSM interpreter can have extended functionality related to different applications. This includes, for example, special algorithms for role negotiation and PaP support functionality (e.g., the use of ActorPlugIn, RoleSessionAction and other APIs). The FSM interpreter can be implemented by Java, C++ and other programming languages depending on the platform used, which may be TAPAS, ServiceFrame [7] and others.

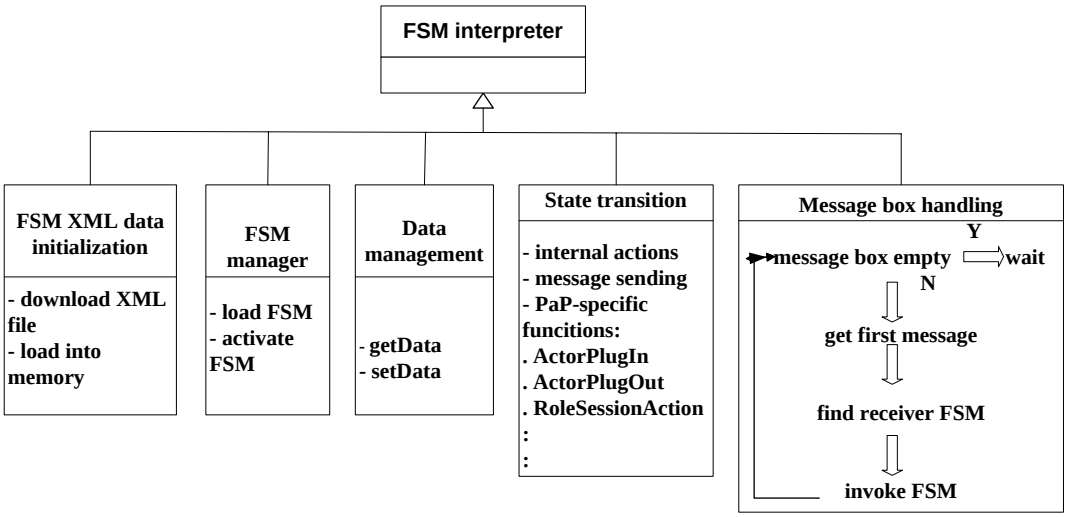


Figure 4. Functionality for FSM interpreter

The XML file has a simple and well-extensible data structure. It is implementation language independent. The XML file consists of all the basic elements for state machines (Figure 5). The set of states is defined together with initial state and current state. The transition table is used to make transitions according to the current state and the input signal/message. Application specific data is also recorded in the XML file.

Internal actions that are carried out during a state transition are defined with an <action> list. This XML structure has good extensibility and allows for easily adding new abstractions. For example, if method interface is desirable for the EFSM, the method signatures can be represented in a similar structure as <action> list.

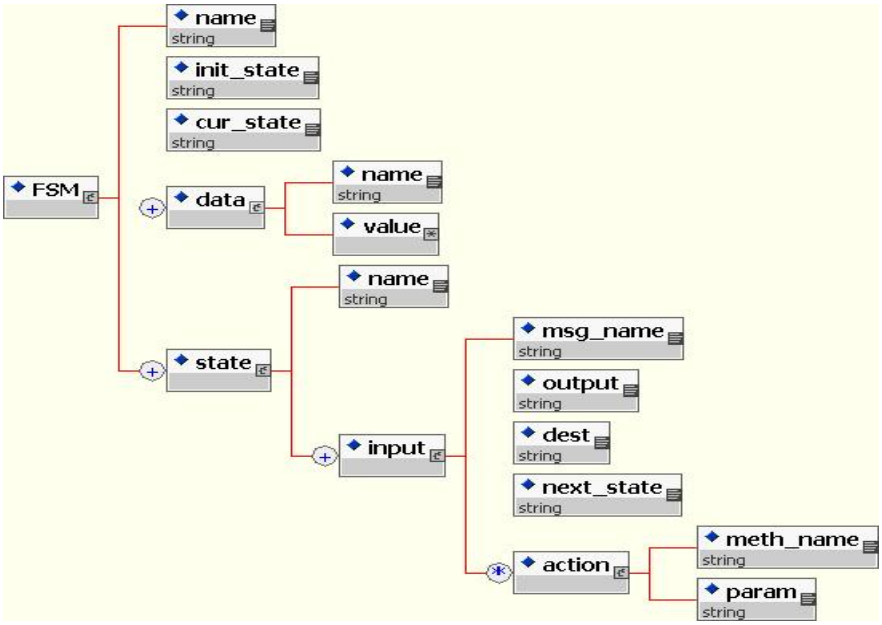


Figure 5. Basic XML data structure

Different FSM Interpreters can be implemented to exchange XML-based behaviour description in different systems. XMI (XML Metadata Interchange) [10,11] is a widely used exchange format, which is also based on XML. XMI is an OMG Standard, and a model driven XML Integration framework for defining, interchanging, manipulating and integrating XML data and objects. "The main purpose of XMI is to enable easy exchange of metadata between modelling tools (based on the OMG UML) and metadata repositories (OMG MOF based) in distributed heterogeneous environments [10]." In short, XMI is UML (Object Technology) with XML (exchanging data over Internet). XMI is currently being adopted by a lot of UML Case Tools vendors and used to tie all tools together. For example, using XMI Toolkit from IBM, developers can share Java objects using XML, generate document type definitions (DTDs), and convert designs and code between Java, UML, and the Rational Rose visual modelling tool [12]. To utilise the benefits from XMI, there can be part of the extended functionality for FSM Interpreter to translate XML file into XMI format and XMI to XML for interchange between tools. Since there is no inherent confliction between XMI and the XML-based EFSM file as they have common basis on XML, this conversion functionality can be easily implemented.

4. The Implementation in Java and TAPAS Platform

An example for this XML-based behaviour representation has been implemented on TAPAS platform using Java technology. The reason for using Java is that Java has

many strong points for XML development. It is a mature technology, highly portable and supported by many vendors. Many XML tools and high-quality development environments are also available in Java.

The implementation framework is given in Figure 6. The manuscripts are defined in XML. An FSM interpreter is written in Java to interpret the XML files to work with the Java-based TAPAS platform. The role figures are represented as FSM instances. The FSM interpreter manages all the FSM instances in the memory and handles a common message box for all these FSMs. The manuscripts in XML are downloaded from the Web server during plug-in phase and loaded into memory as FSM instance by FSM Interpreter. FSM interpreter retrieves the first message from the common message box and activates the corresponding FSM instance to make transitions according to the transition table and the received message.

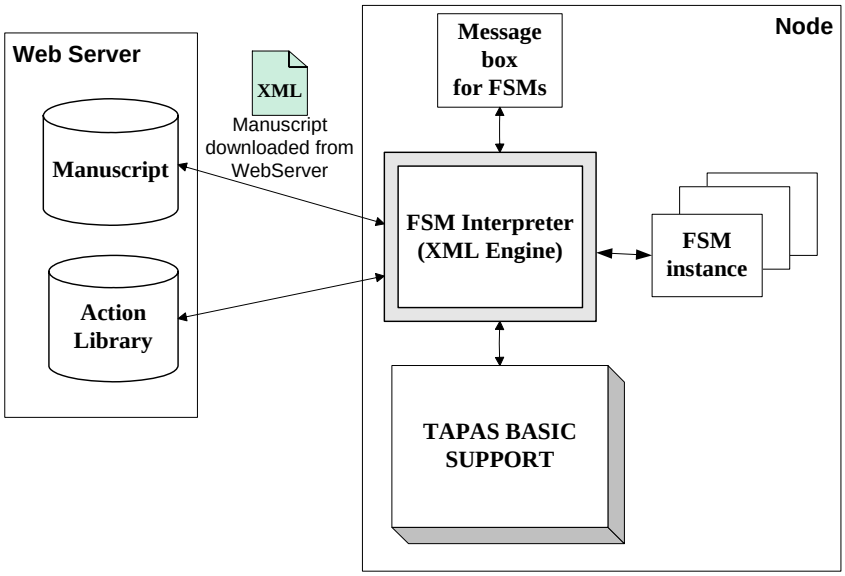


Figure 6. Implementation framework (engineering model)

In XML file, only method name and parameters for each action are specified in an <action> list (cf. Figure 5). The actual actions are implemented in Action Library depending on the underlying platform and language. They are also utilising node inherent capabilities, like printing and file server functions. The application specific data are recorded in FSM instances and real parameter values are provided by FSM instances during runtime. By utilizing Java Reflection [13], each time the FSM interpreter interprets the internal actions, it will invoke the implementation method in Action Library according to the method name and parameters specified in the manuscript. The description and implementation of actions can therefore be separated. Manuscripts and Action Library are created by application designer and are available from the Web server.

Actions include general-purpose actions, such as, printing, encryption, WindowsNew, WindowsClose, and UserLogOn; and application specific actions, such as, setFsmData. Engineering ontology of actions is, therefore, important for implementation of Action Library. However, instead of defining a new ontology for every service system, a standard and predefined one can be shared and reused if it is

available. For example, the process ontology defined in DAML-S [15] is a good candidate. Action library can then be implemented in Java or any other language according to the ontology.

Figure 7 gives an example, which illustrates the structure of the extended TAPAS platform. XML Manuscripts, Action Library, and TAPAS Support System are available from Web-server. The Actor-environment-execution-module (AEEM) is a process or thread that executes a collection of actors with associated Plug-and-Play Actor Support (PAS). FSM interpreter manages the Actors instantiated as FSM instances on each node. Director and the Actors which the director manages can reside on different nodes.

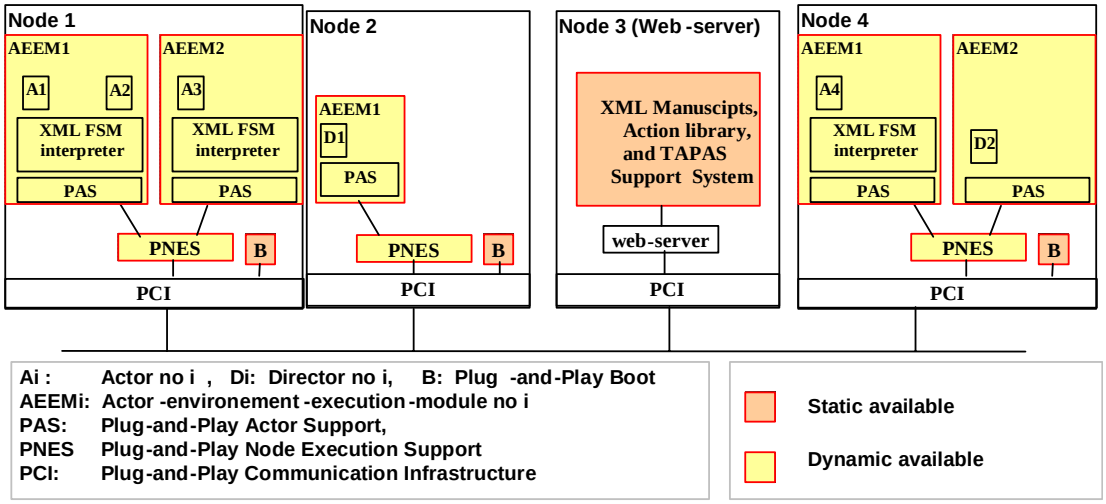


Figure 7. Example view of extended TAPAS platform for software execution

5. Conclusion

Modelling of behaviour is important for dynamic adaptable service development and deployment. An implementation language independent framework for behaviour description in XML is presented in this paper. This model is based on an FSM interpreter (XML engine), which is the interface between XML-represented behaviour description and runtime environment. FSM interpreter can be implemented in different languages and platforms, therefore, the model presented in this paper provides a solution for different systems to interoperate with each other and interchange data and behaviour. By defining or applying action ontology, an Action Library can be built for underlying platform, which utilises the node inherent capabilities and makes the XML-based behaviour description platform independent. The model has been implemented and integrated into TAPAS platform. Conversion between XML file and XMI can be provided as part of the functionality of the FSM Interpreter to make use of the benefits of XMI.

The result for the framework presented in this paper is a common XML representation for i) Dynamic service configuration and ii) Dynamic service instantiation. The XML-based behaviour representation presented here will also be used in mobility handling architecture for actor movement. When an Actor is going to move from one node to another, all the FSM instance data will be packed in an XML file

using the same structure as Figure 5, and sent to destination, where it will be unpacked by FSM interpreter so that the Actor can resume execution in the new node.

Service behaviour is not only procedures. For further perspective, the XML description makes it possible to reason behaviours based on some XML reasoning engine, such as an XET reasoning engine [16]. As an application for this, consider the interoperation between service systems based on TAPAS and other architectures. When a service request is given, there must be a dynamic mechanism for service discovery to match the required and provided services. The framework presented here can be used to reason between the service requirements and behaviours provided by the system based on the XML behaviour representation during service discovery.

References

1 Aagesen, F. A., Helvik, B., Johansen, U. and Meling, H. (2001) Plug and Play for Telecommunication Functionality: Architecture and Demonstration Issues. *Int'l Conf. Information Technology for the New Millennium* (*IConIT*), Thammasat University, Bangkok, Thailand.

2 Aagesen, F. A., Helvik, B., Anutariya, C., and Shiaa, M. M.(2003) On Adaptable Networking. *ICT'03*. April 2003, Bangkok, Thailand.

3 Aagesen, F. A., Helvik, B., Wuvongse, V., Meling, H., Bræk, R. and Johansen, U. (1999) Towards a Plug and Play Architecture for Telecommunications. *Proc. 5th IFIP Conf. Intelligence in Networks* (*SmartNet'99*), Bangkok, Thailand. Kluwer Academic Publisher.

4 Aagesen, F. A., Anutariya, C., Shiaa, M. M. and Helvik, B. E. (2002) Support Specification and Selection in TAPAS. *Proc. IFIP WG6.7 Workshop and EUNICE Summer School on Adaptable Networks and Teleservices*, Trondheim, Norway, September, 109-116.

5 Shiaa, M., M. Mobility Support Framework in Adaptable Service Architecture. Network Control and Engineering for QoS, Security and Mobility 2003 IFIP/IEEE Conference (NetCon'2003), Muscat-Oman, October 2003.

6 W3C: eXtensible Markup Language (XML): http://www.w3.org/XML/.[June 12, 2003]

7 Bræk, R., Husa, K., E., Melby, G. ServiceFrame Whitepaper, *Ericsson NorARC*, May 2002.

8 SuperX++. http://xplusplus.sourceforge.net/index.htm [June 12, 2003]

9 Florescu, D., Grünhagen, A., Kossmann, D. *XL: An XML Programming Language for Web Service Specification and Composition* .WWW2002, International World Wide Web Conference, Honolulu, HI, USA, May 7-11, 2002.

10 Object Management Group. XML Metadata Interchange, version 1.1. ftp://ftp.omg.org/pub/docs/ad/99-10-02.pdf.

11 Nguyen, Hai Thanh. XML Based Data Exchange: Requirements and evaluation of XMI, GML and ISO 19118. Cand Scient Thesis. University of Oslo, 2001.

12 IBM developerWorks. Convert designs and code between Java, UML and Rational Rose. June 1, 2001. http://www-106.ibm.com/developerworks/webservices/library/co-cow21.html [June 12, 2003]

13 Sun Microsystems, inc. Java Core Reflection. http://java.sun.com/j2se/1.3/docs/guide/reflection/spec/java-reflectionTOC.doc.html [June 12, 2003]
14 Wuwongse, V., Anutariya, C., Akama, K. And Nantajeewarawat, E. (2001) XML Declarative Description (XDD): A language for the Semantic Web. *IEEE Intelligent Systems* 16(3):54-65.
15 The DAML Services Coalition. DAML-S: Semantic Markup for Web Services. http://www.daml.org/services/daml-s/0.7/daml-s.pdf. [June 12, 2003]
16 Anutariya, C., Wuwongse, V. And Wattanapailin, V. (2002) An Equivalent-Transformation-Based XML Rule Language. *Proc. Int'l Workshop Rule Markup Languages for Business Rules in the Semantic Web*, Sardinia, Italy.

PAPER E: Automatic Translation of Service Specification to a Behavioural Type Language for Dynamic Service Verification

Shanshan Jiang, Cyril Carrez and Finn Arve Aagesen

Published in
Proceedings of RISE 2004 on Rapid Integration of Software Engineering techniques

Luxembourg, November 26, 2004.

Lecture Notes in Computer Science (LNCS) 3475, pp. 34-44, 2005.

Automatic Translation of Service Specification to a Behavioural Type Language for Dynamic Service Verification

Shanshan Jiang

Cyril Carrez

Finn Arve Aagesen

Abstract Networked *services*, constituted by the structural and behaviour arrangement of *service components* are considered. A service component is executed as a *generic software component*, denoted as an *actor*, which is able to download and execute different EFSM (Extended Finite State Machine) based functionality. The functionality of an actor is denoted as its *role*, while a *role session* is a projection of the role with respect to the interaction with one other actor. We propose an approach for verification of the services, based on interface verification techniques for the verification of the role sessions. The service component specifications used for actor execution are based on XML representations, while the verification of the role sessions is based on a *behaviour type language*. This language has a sound theoretical basis, and is used to avoid "message-not-understood" errors. Rules are given for automatic translation from the XML manuscripts to this behavioural type language. This translation first makes projection to the role session, using hidden actions. Those hidden actions are then removed so a sound verification can take place.

1. Introduction

Networked *services* constituted by *service components* are considered. A service component is executed as a software component in *nodes*, which are physical network processing units such as servers, routers or switches, and user terminals such as phones, laptops and PDAs. Traditionally, the nodes and the service components have a predefined functionality. Concerning both the nature of nodes and the software engineering principles, changes are taking place. From being a static component, the service component can be based on generic software components, which are able to download and execute different functionality depending on the need. Such generic programs are from now on denoted as *actors* (by analogy with the actor in the theatre). The functionality of an actor is denoted as its *role*, while a *role session* is a projection of the role with respect to the interaction with one other actor. *Role* and *role session* need the power of Extended Finite State Machines (EFSMs) in order to describe a complex protocol of interaction (which is not the case with most Web-based services as they are request-reply services specified as procedure calls).

There are basically two different approaches to service verification. One is to model the composite behaviour of the whole service [Hol90], but it leads to state explosion and has limited applicability for complex systems. Another approach is the decomposition of the service system and isolated verification of the decomposed parts (new services

that reuse existing components can take full advantage of this compositional verification). Within this approach we have the sub-approach focusing on the interfaces between the service components, with interface type languages [CFN05, LX04]. Most of these approaches use behavioural type systems [Nie95, NNS99,RV00], where a type specifies a *non-uniform* interface, meaning the set of operations (or messages) the interface accepts depends on the context. Indeed, this type is viewed as an abstract behaviour of the component, and is used during compositional verification to ensure liveness and safety properties of the application. We aim for a quick compositional verification, restricted to the compatibility verification of connected interfaces. This will keep the number of states very low, instead of verifying the compatibility of the whole behavior of the components. We used the type language developed in [CFN05], preferred because of its high level of abstraction. In this setting, each component must satisfy a contract, which specifies the behavioural type of its interfaces; an assembly of components is sound if connected interfaces are compatible.

This work is part of the TAPAS project (*Telematics Architecture for Play-based Adaptable System*), which goal is to enhance the flexibility, efficiency and simplicity of system deployment, operation and management by enabling dynamic configuration of network-based service functionality. See [AHAS03] and the URL http://tapas.item.ntnu.no. We propose an approach to verify the services, based on interface verification techniques for the verification of the role sessions. We provide an automatic translation from XML-based EFSM service component specification to the behavioural type language applied. The projection process has two steps. First, make projections that preserve the binding between the role sessions related to each service component by using hidden actions. Then remove the hidden actions so a sound verification can take place.

The paper is organized as follows. The context of our service specification is described through the related TAPAS concepts (Sect. 2). The behavioural type language used in the verification is described in Sect. 3. Section 4 gives the methodology of translation. Related work and Conclusion end the paper.

2. Some TAPAS Concepts

Part of the TAPAS architecture relevant for the verification is illustrated in Figure 1. For a more comprehensive description of the architecture, see [AHAS03]. Concepts such as service, service components, actor, role and role session were defined in Section 1. The concepts of actor, director, role and manuscript are concepts from the theatre, where actors play roles according to manuscripts and a director manages their operations. An actor has two kinds of interfaces (Fig. 2):

Home Interface. This is a control interface between an actor and its director. Each actor is associated with one Director, who manages the performance of the actor through this control interface.

Application Interface. This is an interface where the role sessions between actors take place.

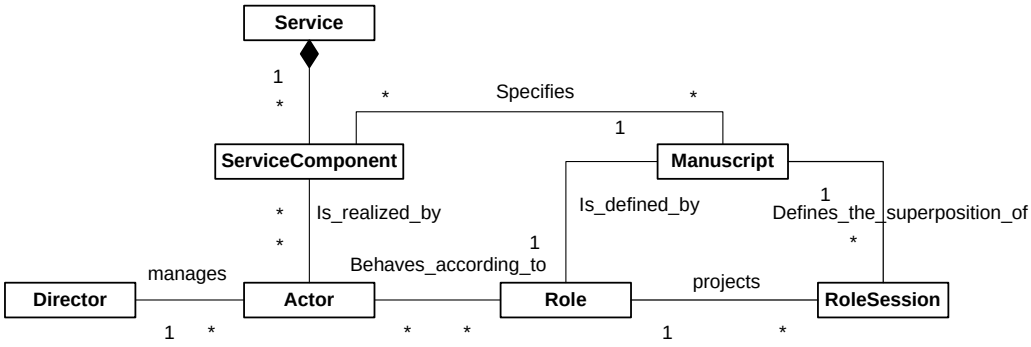


Fig. 1. Some TAPAS concepts related to the verification

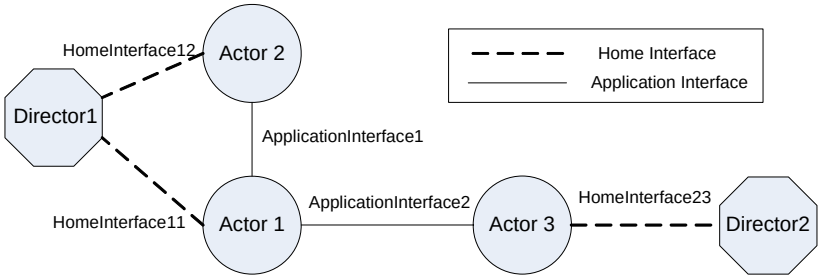


Fig. 2. Actor interfaces

Figure 3 shows the basic data structure of an XML manuscript. This manuscript is the specification of the EFSM based behaviour of an actor. It contains the name of the EFSM, its initial state, data and variables, and a set of states. The state structure defines the name of the state and a set of transition rules for this state. Each transition rule specifies that for each input, the EFSM will perform a number of actions, and/or send a number of outputs, and go to the next state. The actions are functions and tasks performed during a specific state: calculations on local data, method calls, time measurements, etc. The <actions> list specifies only the action type (method name), parameters and action group (the classification of action types). This XML manuscript therefore specifies parameterized behavioural patterns. The detailed platform support and example implementation for XML service specification can be found in [JA03].

A fragment of an example XML manuscript is given in Fig. 4. This fragment comes from an example application called TeleSchool, which we used as an experiment of our approach. For lack of space, we only show simplified behaviour description for one state. This service component specification will serve as example to demonstrate the translation rules later.

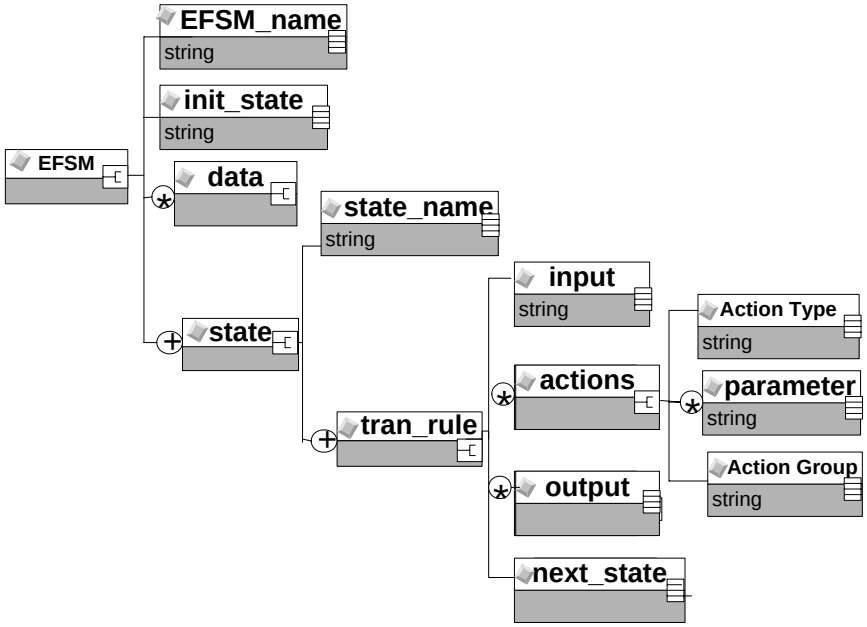


Fig. 3. Manuscript data structure

```
<state name="stInitUserInterface">
  <input msg="LogonEventInd"
         source="v_interface">
    <actions>
      <ActionType>ActorPlugIn
      </ActionType>
      <param>
        <name>role</name>
        <value>SchoolServer</value>
      </param>
      <ActionGroup>G1</ActionGroup>
      <store_return>v_server
      </store_return>
    </actions>
    <actions>
      <ActionType>setVariable
      </ActionType>
      <param>
        <name>value</name>
        <value>INPUT_MSG.school
        </value>
      </param>
      <ActionGroup>G2</ActionGroup>
      <store_result>v_currentSchool
      </store_result>
    </actions>
    <actions>
      <ActionType>setVariable
      </ActionType>
      <param>
        <name>value</name>
        <value>INPUT_MSG.user</value>
      </param>
      <ActionGroup>G2</ActionGroup>
      <store_result>v_currentUser
      </store_result>
    </actions>
    <output>
      <msg type="UserVerifyAccessReq">
        <param>
          <name>message</name>
          <value>INPUT_MSG</value>
        </param>
        <dest>v_server</dest>
      </msg>
    </output>
    <next_state>stPasswordIdentify
    </next_state>
  </input>
  <input msg="CancelEventInd"
         source=...>
    <actions>...</actions>
    <next_state>stInit</next_state>
  </input>
</state>
```

Fig. 4. Fragment of an example XML manuscript

3. Behavioural Type Language

We adopt the behavioural type language introduced in [CFN05]. This language describes messages that are exchanged on interfaces. We chose this language because it has a well defined semantics, and is based on a component model which is rather close to the Actor model of TAPAS. A component in [CFN05] has a set of ports. Each port interacts with a so-called *partner*, with which it sends and/or receives messages. Communication is asynchronous, and is made through an abstract communication medium containing FIFO queues (one for each port). A port will then be mapped to the interface in TAPAS, the main difference being that each port has its own queue, whereas in TAPAS there is one queue for the whole component. However, we think the two models are equivalent: retrieving, in a global queue, a message destined to an interface is similar to picking up the first message in the queue of that interface. The strong formal framework of the language in [CFN05] also allows us to avoid "message-not-understood" errors, and to ensure external deadlock freeness properties[1]. Moreover, a type not only imposes constraint on the interface it specifies, but also on its environment: it is possible to enforce the environment to send a message by specifying that the interface "**must** receive a message". Although this feature has not been used yet, we think it has an important impact on liveness properties when composing services.

In this paper the details of this language are not presented; the interested reader should consult [CFN05, CFN03], where a BNF table is developed, as well as semantics description and examples. However, we present an example of a bank account specification. The following type specifies the *operations* interface through which a client might perform credits and withdrawals:

operations = **may ?[** deposit (real)**; must ! [** balance (real)**;** operations **]**
+ withdraw (real)**; must![** balance (real)**;** operations
+neg_balance (real)**;**negbal_operations**]]**
negbal_operations = **must ? [** deposit (real)**;**. . .+ withdraw (real)**;**. . . **]**

This type is read as follows: *operations* may receive (**may?**) *deposit* and *withdraw* messages. After receiving one of the two messages, the interface must send (**must!**) the balance of the bank account: message *balance* is sent when balance is positive, and the type becomes *operations* again. Message *neg_balance* is sent when the user is debtor, and then the type becomes *negbal_operations*. This latter type is similar to the *operations* type, with the exception of the modality: type *operations* **may** receive messages, whereas *negbal_operations* **must** receive messages. Hence, the client must perform some operations as long as he is debtor.

For the time being, we concentrate on the choice operator "+" and the sequence operator "**;**", so the resulting type is an abstract behaviour of the component, which is roughly a projection of its behaviour to a specific interface.

[1] The "message-not-understood" error avoidance is mainly due to the compatibility of the types of the interfaces. The deadlock freeness property is due to constraints on the internal behaviour of the components, mainly on dependencies between interfaces (i.e. an interface waiting for a result on another one).

4. Translation Methodology

The actions that an actor can perform are classified into three types:

Control functionality through Home Interface: management functionality including `ActorPlugIn, ActorPlugOut`, etc., defined in [JAHB99]. A request is sent to the Director and the Actor must wait for the result.

Role session through Application Interfaces: the application interactions use asynchronous message sending through Application Interfaces.

Internal actions: they are invisible in the interface descriptions.

The translation from service specification to interface language uses projection. Projection is an abstraction technique, which can produce a simplified system description or viewpoint by aggregating some of the system's functionalities while hiding others. It has been used in previous works [LS84, Flo03] to simplify the verification of protocols and validation analysis. In our approach, the projection process basically consists of two steps. The first step extracts the inputs and outputs for a specific interface. All other actions are considered as hidden (internal actions and interactions occurring at other interfaces). The second step removes those hidden actions so a sound verification can take place.

The automatic translation algorithm is as follows. It scans the XML manuscript once and extracts the interface interaction information, the translation procedure being carried out state by state. Each interface has a unique identifier (for example `I1`), which is assigned dynamically when the interface is created. For each interface, every state has a type name assigned, which is composed of the interface identifier and a number. For example, interface types `I1_*` are used for all the interactions with the HomeInterface (Director), where `I1_1` is the type of the first interaction, which will be transformed to `I1_2` after some I/O interaction. This dynamic creation of interface type is flexible and easy for implementation. Each behaviour description will be translated to the equivalent interface type description (using the translation rules described hereafter), affecting one interface at a time. Finally, gathering of silent transitions and states (Sect. 4.3) is applied on the interface type descriptions. In order to simplify the translation, each state will receive only messages from one single interface (i.e. all the inputs for one state are from the same source). If inputs are from different interfaces, we create new states for processing them. In our first implementation, **must** and **may** modalities are not distinguished: all actions will be "**must**".

4.1 Messages

In TAPAS, all communications are through asynchronous message passing. The input and output operations are the visible actions through interfaces and are translated directly into interface types. An input message means a receiving interface type "**?**", while an output message means a sending interface type "**!**". Synchronous communication can be implemented by an output followed by an input message, thus translated into a sending interface type "**!**" followed by a receiving one "**?**".

We consider only the types of the parameters, not their values. The resulting message types are then finite, so the validation will be a finite-state verification (our verification

is an optimistic one: it does not handle the cases where values are received outside the scope of their type).

The XML message structures are input, output, and control functionality.

<input> structure. The parameter "source" identifies the role session, and distinguishes the interface for this input operation. This structure is translated to "**?**M_i", with M_i the message type.

Example 1.

XML manuscript	Interface type
<**input** msg="LogonEventInd" source="v_interface" >	I2_2 = **must ? [** LogonEventInd**;** I2_3 **]** *I*2_∗ is used for the interface "v_interface"

<output> structure. The parameter "dest" identifies the role session for this output operation, and is used to find the binding interface. If it represents a new destination, a new interface will be created. This structure is translated to "**!**M_i", with M_i the message type.

Example 2.

XML manuscript	Interface type
<**output**> <**msg** type="UserVerifyAccessReq"> <**param**> <**name**>message</**name**> <**value**>INPUT_MSG</**value**> </**param**> <**dest**>v_server</**dest**> </**msg**> </**output**>	I3_1 = **must! [**UserVerifyAccessReq**;**I3_2**]** *I*3_∗ is used for the interface "v_server"

Control Functionality in <actions> Structure. This is identified by a method name starting with "Actor" in <ActionType> substructure. This should be translated to "**!** M_i**;** **?** M_j" for the HomeInterface.

Example 3.

XML manuscript:

```
<actions>
  <ActionType>ActorPlugIn</ActionType>
  <param>
    <name>role</name>
    <value>SchoolServer</value>
  </param>
  <ActionGroup>G1</ActionGroup>
  <store_return>v_server</store_return>        <!–of type "roleSessionId"–>
</actions>
```

Interface type:

I1_2 = **must! [** ActorPlugIn (role)**; must?[** RequestResult(roleSessionId)**;**I1_3**]]**
*I*1_∗ is used for the interface "HomeInterface"

4.2 Deactivation of Interfaces

Interfaces can be dynamically created and deleted (deactivated). Deactivation is reflected by the "ActorPlugOut" control functionality. It plugs out an interacting actor, thus placing the corresponding role session into inactive state "**0**". The interface can then be deleted, the deletion being an internal behaviour.

4.3 Hidden Actions and Their Removal

The first step of our projection on one interface replaces internal actions and interactions occurring at other interfaces by hidden actions, also called τ -transitions. The next step in the projection is to remove those hidden actions, by combining τ -transitions and states, as in Floch's work [Flo03].

All the successive τ -transitions can be replaced by one single τ -transition, and the states are combined into one state, as shown in Fig. 5(b). If input and output sequences are the same for two states, these states may be combined. However, some ambiguous behaviour may result, as shown in Fig. 5(c): from state 2' message *M*2 or *M*3 can be received, while originally *M*2 can be received only in state 6, and *M*3 in state 5 (This ambiguity may be due to some hidden parameter values, or to dependency between interfaces). We eliminate this ambiguity by adding state information in the name of the message (Fig. 5(d)): "**!** *M*1{*S*4}" means "output a message type *M*1 at the state *S*4". The final translated type for Fig. 5(a) could be expressed as follows, referring to Fig. 5(d):

I1 = **must ! [** M1{S4}**; must ? [** M2**;** I7**]** + M1{S3}**; must ? [** M3**;** I7 **]]**
*I*1 and *I*7 are the interface type description for State1 and State7 respectively.

As states and input types (messages) are finite, our types have finite states, thus avoiding the infinite state verification problem. We also provide a more accurate description of interface behaviour than the traditional interface definitions, which specifies signature of methods but not complex interface behaviour.

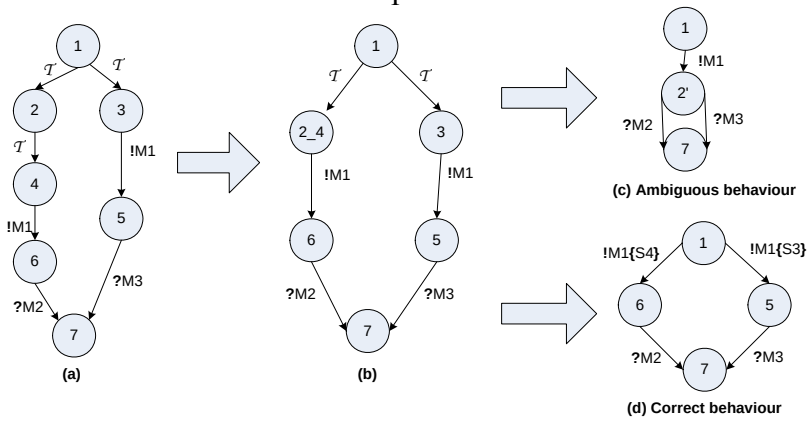


Fig. 5. Combination of τ – transitions

5. Related Work

Floch's PhD thesis [Flo03] provides a validation approach for dynamic service composition, which is similar to our work. Floch models the behaviour of the service components as state machines using SDL; projection is used to transform it into

interface behaviour (also described as state machines using SDL-like notation). Our service specification has a higher level abstraction of behaviour, as only action types are defined. Therefore, implementation details of the internal actions are already hidden, while at the same time keeping all the information about interface interactions. This simplifies the translation process. Another difference is that we provide translation by directly analyzing the XML data structure, whereas the transformation in Floch's work is based on state graphs.

The behavioural type language we used was first issued in [CFN03], and further developed in [Car03, CFN05]. Many type systems exist to capture the behaviour of processes, actors or components, most of them based on process algebras like π-calculus. The closest type system to the one we used is the one of Najm et al. [NNS99]. The authors propose an actor calculus featuring regular or infinite state behaviour. Although they detect "message-not-understood" errors, communications are one-way. Ravara and Vasconcelos, in [RV00], were also inspired by Najm et al., but they did not make any distinction between inputs and outputs, hence their notion of error is rather loose and did not fit in our needs. They corrected this with Gay, providing so-called "session types" [GVR02], but distinguishing internal and external choice between actions (respectively client and server choices), which means we have to add messages to make sure those choices are made accordingly. De Alfaro and Henzinger provide another way of specifying interface behaviour, using Interface Automata [dAH01]. Those automata specify the sequence of input/output allowed on an interface, but the compatibility they develop is too weak for our architecture. Indeed, two components are compatible if there exists an environment that can interact with the product automaton of the components' types; we believe this does not allow detecting "message-not-understood" in a plug-and-play environment such as TAPAS.

6. Conclusion

We have presented an approach for verification of the services, based on interface verification techniques for the verification of role sessions. We also provide an automatic translation from XML-based EFSM service specification to the behavioural type language applied. This language has a sound theoretical basis, and provides formal framework for compositional verification of component based systems. Especially, it allows us to capture "message-not-understood" errors while plugging a new component. The automatic translation provides an efficient and reliable way to extract interface types. An experiment has been carried out on an example application called TeleSchool.

For further work, the translated interface description can be used to compare specifications of behaviour / service, so that dynamic service discovery can be done based on more accurate semantic interface behaviour description comparison instead of simply signature matching. Furthermore, the dynamic assembly of components for validation needs to be further developed so as to provide safe plug-and-play techniques for components. Finally, we did not use all the features provided by the behavioural type language, and put aside the **may** and **must** modalities on the actions. We think about using the latter modality ("I have to send/receive") together with service-goal developed by Sanders and Bræk [SB04]: some actions can be specified as obligatory (**must**), so the service goal is fulfilled.

References

[AHAS03] F. A. Aagesen, B. E. Helvik, C. Anutariya, and M. M. Shiaa. On adaptable networking. In *ICT'03, Proceedings*, Assumption University, Thailand,2003.

[Car03] C. Carrez. *Contrats Comportementaux pour Composants.* PhD thesis, ENST, Paris, France, December 2003.

[CFN03] C. Carrez, A. Fantechi, and E. Najm. Behavioural contracts for a sound composition of components. In *FORTE'03*, volume 2767 of *LNCS*. 2003.

[CFN05] C. Carrez, A. Fantechi, and E. Najm. Assembling components with behavioural contracts. *Annals of Telecomms*, 2005. To appear. Ext. of [CFN03].

[dAH01] L. de Alfaro and T. A. Henzinger. Interface automata. In *ESEC/FSE-01*, volume 26, 5 of *Software Engineering Notes*. ACM Press, 2001.

[Flo03] J. Floch. *Towards Plug-and-Play Services: Design and Validation using Roles*. PhD thesis, NTNU, Trondheim, Norway, February 2003.

[GVR02] S. Gay, V. T. Vasconcelos, and A. Ravara. Session types for inter-process communication. Preprint, Dept. of Computer Science, Univ. of Lisbon, 2002.

[Hol90] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, November 1990.

[JA03] S. Jiang and F. A. Aagesen. XML-based dynamic service behaviour representation. In *NIK'03, Proceedings*, Oslo, Norway, Nov. 2003.

[JAHB99] U. Johansen, F. A. Aagesen, B. E. Helvik., and R. Bræk. Design specificationof the PaP support functionality. Technical Report 1999-12-10, Department of Telematics, NTNU, 1999.

[LS84] S. S. Lam and A. U. Shankar. Protocol verification via projections. *IEEE Transactions on Software Engineering*, 10(4):325–342, July 1984.

[LX04] E. A. Lee and Y. Xiong. A behavioral type system and its application in ptolemy ii. *Formal Aspects of Computing*, 16(3):210–237, August 2004.

[Nie95] O. Nierstrasz. Regular types for active objects. In *Object-Oriented Software Composition*, pages 99–121. Prentice-Hall, 1995.

[NNS99] E. Najm, A. Nimour, and J.-B. Stefani. Infinite types for distributed objects interfaces. In *FMOODS'99, Proceedings*, Firenze, Italy, February 1999.

[RV00] A. Ravara and V. T. Vasconcelos. Typing non-uniform concurrent objects. In *CONCUR 2000*, volume 1877 of *LNCS*, pages 474–488. Springer, 2000.

[SB04] R. Sanders and R. Bræk. Discovering service opportunities by evaluating service goals. In *EUNICE'04, Proceedings*, Tampere, Finland, June 2004.

PAPER F: An Approach for Dynamic Service Management

Shanshan Jiang, Mazen Malek Shiaa and Finn Arve Aagesen

Published in
Proceedings of IFIP WG 6.3 Workshop and EUNICE 2004 on "Advances in fixed and mobile networks"

Tampere, Finland, June 14-16, 2004.

An Approach for Dynamic Service Management

Shanshan Jiang

Mazen Malek Shiaa

Finn Arve Aagesen

Abstract Managing dynamic changes in a service system requires the handling of the modifications and extensions of the system without stopping or disturbing its functionality. For this purpose, a framework for dynamic service management with respect to service specification, selection and adaptation is proposed. The service specification is based on modifiable and parameterised behaviour patterns, which can be instantiated by dynamically selecting the execution codes according to the execution environment. The service adaptation is achieved by allowing these specifications to be modified according to the dynamic changes in the executing environment. A prototype selection engine has been implemented to demonstrate the feasibility of the framework, and an example that exploits these features is presented.

Index terms: Plug-and-play systems, Dynamic Service Management, Service Adaptation.

1. Introduction

Dynamic Service Management is the mechanism of changing the configuration and performance of the service system during execution. A service system is viewed as a composition of service components. The problem area of Dynamic Service Management may be divided into three main issues:

- *Service Specification*: How can a service be specified so that it is possible to manage and cope with dynamic changes without major human interventions?
- *Service Selection*: If there exist multiple specifications that may be instantiated to realize a service component, how to select the most appropriate one given certain information on the operating environment?
- *Service Adaptation*: What procedures and routines can enhance the service adaptation to the dynamic changes in its execution environment related to resources, users and changed service requirements?

In this paper, we propose an approach for Dynamic Service Management. The main idea is to base the *Service Specification* on behaviour patterns (basically state machine specifications) with generalized action types, while the actual executing code, or the action libraries that include routines specific to the execution environment, is determined during run-time. This selection of optimal code for execution is referred to as *Service Selection*, and is based on the execution environment context, and of course the service request. *Service Adaptation* is achieved by allowing the service components

to modify their functionality, or the code they run, dynamically by requesting changes to their service specification. The Selection and Adaptation processes are solely influenced by the information related to system resources, which are represented by *capability and status* information. To obtain a general sight of the rationale behind our approach, consider the following example, a "server" that provides some sort of a service to varying number of users, resides in a changing environment in terms of available system resources. Now, by the realization of different scenarios, keeping these changing factors in mind, this server may benefit from an adapting service specification, e.g. whether to use local data space or database utilities to manage users' data. The desirable situation is that the server can automatically switch to different operating modes, for example, when the number of users changes or the node it is operating within undergoes a congested traffic. The major concern here is the limited and varying resources in the system, as well as the need to provide smooth operating principles for certain executing service components.

The concept and framework presented in this paper has been worked out within TAPAS (Telematics Architecture for Plug-And-play Systems), which is a research project aims for developing an architecture for adaptable network-based service systems. The goal is to enhance the flexibility, efficiency and simplicity of system installation, deployment, and management by enabling dynamic configuration of network components and network-based service functionality. See [1,2,4,5] and the URL: http://tapas.item.ntnu.no.

Sec. 2 will give a general overview of some related work that share similar vision to our work. Sec. 3 will provide some details of the concept of service and capability as viewed in TAPAS. Sec. 4 will present the TAPAS core platform, while Sec.5 will handle the proposed framework and work out the various topics that constitute the overall solution. Sec.6 shows a use-case example of the framework, and finally Sec. 7 provides some concluding remarks.

2. Related Work

Network-based Services have been an important research topic for many industrial and academic bodies. Some examples may be listed here. TINA (Telecommunication Information Networking Architecture) [6] was a major effort in the 90's to provide a solution for the Telecom Service Architecture, including service specification and management. Mobile Agents and Active Networks [7,11,13], and Programmable Networks [12], may also be mentioned as efforts to revolutionalise how services be utilized in an intelligent way. All these topics achieve adaptability in the service provision from different point of views, some focus on the general viewpoints, while others provide support platforms that are closely associated with certain runtime environments. The focus is more and more shifted now towards adaptability and evolution of such services. An example of this changing focus is the IBM autonomic computing project [15].

Service management, to some extent dynamic service management, has been dealt with in a number of papers, e.g. in active networks [10], replacement of software modules in [9]. In general, the principles discussed in [8] are considered to serve as a basis for any dynamic change management, which are also followed in this paper.

Our approach to dynamic service management also focuses on the adaptability of network-based services. We follow basic principles of TINA, yet we try to provide more concrete and concise results. At the same time, TAPAS architecture is based on a generic system, rather than just a particular network, and aims at self-configuration and self-adaptation, which is not solved in Active Networks.

Parlay [18] and JAIN APIs [19] provide service creation environment for rapid creation of telecom services. Our support for rapid service creation is augmented by the consideration and support of adaptation of specification at runtime, which is not directly covered in Parlay or JAIN.

Our service specification based on parameterised behaviour pattern can be seen as an approach to model business process using XML, which has great similarity with Web Services Orchestration approaches such as BPEL4WS [20]. BPEL4WS defines different structures to model process in a manner similar to programming logic, while in our approach, service specification is modelled as EFSM, therefore, implicitly defines the control logic and scheduling by states and message interactions. The great complexity of the service modelled by TAPAS is that we take capability information into consideration. We describe the mechanism for service behaviour adaptation based on the capability information, which is not addressed in orchestration approaches.

3. TAPAS Conceptual Model

The TAPAS conceptual model is illustrated in Figure 1. TAPAS concepts are founded on a theatre metaphor, where *actors* perform *roles* according to predefined *manuscripts,* collectively defined in *plays*. *Actors* are implemented by *software components* in the *nodes* of the network. A *service-system* consists of *service- components,* and is defined by *plays*. *Service-components,* units related to some well-defined functionality, are realised by *role-figures* based on *roles* defined by *manuscripts* and are executed by *actors*. A *role-figure*, however, is realised in an executing environment in a *node* and is utilising *capabilities*. A *capability* is an inherent property of a *capability-component* in some node. A *capability-component* may have several *capabilities*. These *capabilities* are offered to *actors*, which constitute *role-figures* in various *plays*. Basically, capabilities can be classified into:

- *Functions*: pure software or combined software/ hardware components used for performing particular tasks,
- *Resources*: hardware components with finite capacity, such as processing, storage and communication units,
- *Data*: just data, the interpretation, validity and life span of which depend on the context.

Capability Category is a classification of capability sets. Because the functionality provided by a role-figure can be classified into capability dependent functionality and capability independent functionality, *Capability Category* is used in service specification to reflect this dependency on capability.

Status is, at a certain time instant, the situation in a TAPAS system with respect to the actual number of nodes, playing plays, traffic situation, etc. Status can both comprise observable counting measures, measures for QoS or calculated predicates

related to these counts and calculated measures. It reflects the resulting state of the system, which cannot directly be changed and negotiated.

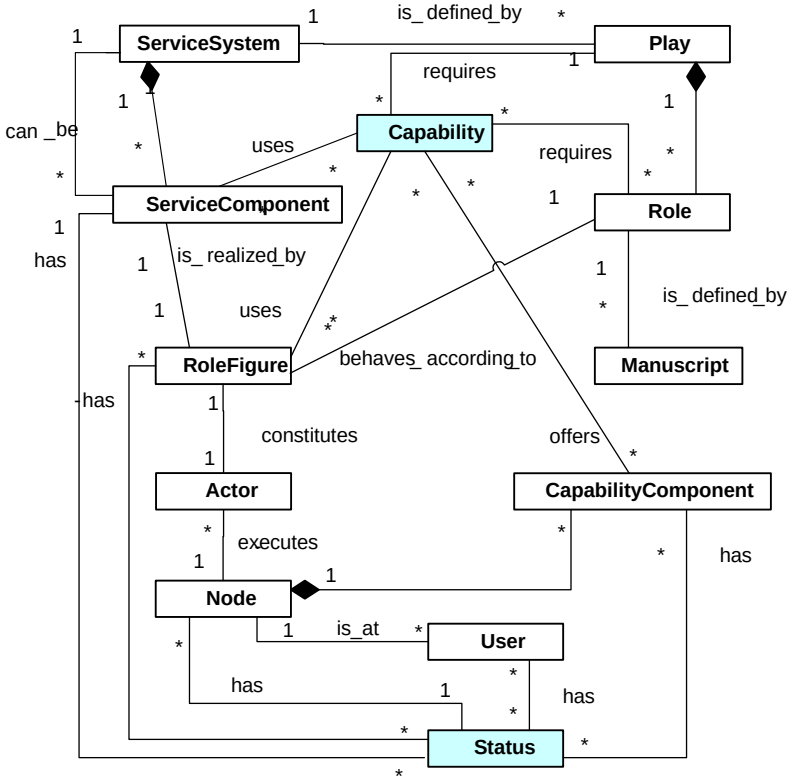


Figure 1 TAPAS conceptual model.

A role-figure executes in some *context,* which may change during its life span. Context information that affects a role-figure's execution can be classified into:

- *Node context*: information about the node a role-figure resides at, which will affect the computation capabilities of the node, such as handheld / desktop / Laptop;
- *Location context*: location information which will affect the functional capabilities in general, such as home / university / mobile;
- *Connection context*: the type of connection which will affect the communication capabilities, such as Dial-Up / LAN / Wireless connection;
- *User context*: is used to classify how different user groups are assigned different access rights, startup application, default capabilities, etc., such as group leader, regular user and administrator.

4. TAPAS Core Platform

A service consists of service components, which are realized by role-figures. A role-figure is implemented by an actor, which executes a manuscript modeled as an Extended Finite State Machine (EFSM). TAPAS core platform is the platform supporting the execution of role figures. XML-based manuscripts are applied to provide

a representation that gives a potential for interoperability [5]. The manuscript data structure is shown in Figure 2. The behaviour description defined in the manuscript consists of states, data and variables, inputs, outputs, and different actions. Actions are functions and tasks performed by the role figure during a specific state. They may include calculations on local data, method calls, time measurements, etc. The <Action> list in manuscripts only specifies the action type, i.e., the method name and parameters for the action. Each action type belongs to some Action Group according to the nature of action. The manuscripts therefore specify parameterized behaviour patterns. The actual implementation of actions is realized by different codes stored in an *Action Library* according to the Capability Category it requires.

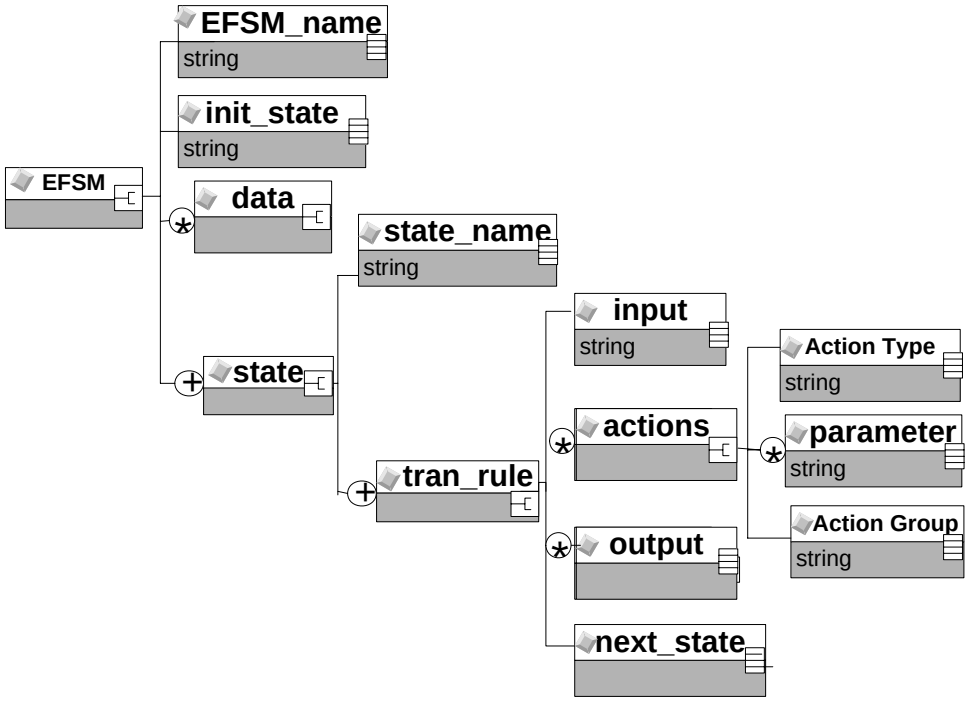


Figure 2 EFSM model data structure**.**

Manuscripts and Action Library, available on a Web server, can be dynamically downloaded to a node and executed by the TAPAS core platform running at the local node. A *State Machine Interpreter (SMI*), which is the primary entity in TAPAS core platform, manages the execution of the manuscripts and the linking of action definitions with their implementation. The SMI executes the manuscript by invoking the action codes with matching action type and capability requirements, i.e., the codes will be selected when the offered Capability Category matches the required Capability Category for the specific action type. This facilitates the separation of action specification and implementation and at the same time provides the flexibility of implementing actions according to capability requirements and programming languages. This mechanism scales as Action Library can be expanded without expanding the manuscript definition and only a subset of Action Library needs to be downloaded.

5. Dynamic Service Management

5.1 The Framework

Figure 3 shows the framework for Dynamic Service Management. Dynamic Service Management is one functionality component of the TAPAS support platform, which extends the Core platform. The components are:

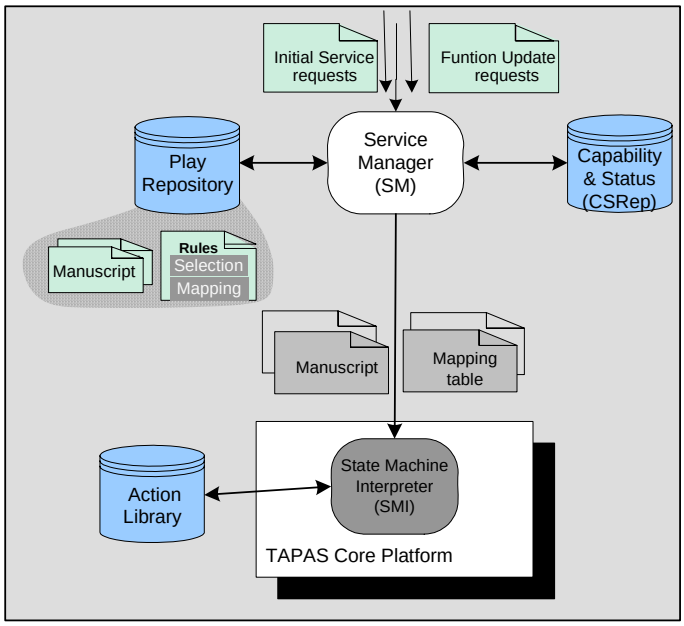


Figure 3 Dynamic Service Management Framework**.**

1. *Play Repository* includes:
   - *Manuscripts* to provide behaviour pattern definition, including the action types to be performed and their corresponding Action Groups.
   - *Rules* to provide information for the decision making process. There are two types of rules, Selection Rules and Mapping Rules. *Selection Rules* are used to dynamically select the proper manuscripts that correspond to a specific role. *Mapping Rules* specify how the context, capability and status information are mapped to capability categories.
2. *Capability and Status Repository (CSRep)* provides a snapshot of the resources of the system. It maintains information on all capability and status data that may affect the execution of the various role-figures at different nodes.
3. *Action Library* contains codes for the state machine-based actions. These codes are implemented according to the capability category they required.
4. *Service Manager (SM)* is responsible for the handling of *Initial Service requests* (to instantiate a service component) and *Function Update requests* (to change the functionality of an already instantiated service component). It, first, selects a specific manuscript based on the *Selection Rules.* Secondly, it calculates the offered Capability Category according to the *Mapping Rules* and generates the *Mapping table*. Both the manuscript and the Mapping table are sent to the *State Machine Interpreter* for execution.

5. *Service requests* supply, on the one hand, the identification for the service component to be instantiated or modified. On the other hand, they will provide the context information that will be taken into consideration during the calculation of the offered Capability Category. Two types of service requests may be handled by the SM:
   - *Initial Service request* indicates a role to be executed at a node.
   - *Function Update request* is issued when the context, available capability or status information has changed and there is a need for functionality update.
6. *Mapping table* is a report that contains the mapping from the Action Groups to the corresponding Capability Categories, which is the result of the calculation done by the SM.
7. *TAPAS Core Platform* provides support for the execution of role figures at a node as discussed in section 4.
8. *State Machine Interpreter (SMI)* is the primary entity in *TAPAS Core Platform* responsible for the execution of manuscripts according to the manuscript and Mapping table sent by the SM.

In the following sub-sections detailed description of these components and their functionality will be given. It is important, however, to notice that the configuration and capability management parts are outside the scope of this paper – their results and consequences are assumed in the Play Repository (the Selection Rules in particular) and the Capability and Status Repository. To obtain a deeper knowledge of these parts of the architecture the reader is requested to visit the project website: http://tapas.item.ntnu.no.

5.2 The Action Library and Capability Category Specifications and Rules

While manuscripts provide parameterised role behaviour pattern specifications, the actual implementation of the actions is realized by different codes in the Action Library. These codes will be designed depending on the capabilities or resources they require or use. Some codes may provide a flexible and powerful functionality, so that they require more resources, e.g. larger memory, faster processor, wideband communication channel, etc. On the other hand, simpler codes, or the so-called lightweight codes, may be aimed for small handheld devices to only support the basic functionality. Allowing the SM to dynamically reason about these different codes based on the availability of system resources would introduce a great degree of flexibility into the manner how the service specification is carried out.

As mentioned in Section 4, actions in the manuscripts are specified as general action types, which all belong to some Action Groups. However, the real code to be executed in order to perform these actions is maintained in the Action Library, which contains the detailed execution code for different executing environments and various operating circumstances, i.e. operating systems, communication infrastructure, etc. The Capability Category, as a way of classifying the capabilities into different sets, is used to indicate which code to be executed. For example, for a certain client role, a Graphical User Interface *must* be obtained; however, the implementation of this interface can be different for different node types that differ in their capability set, e.g. a handheld device or a desktop. Another example is a streaming application that is dependent on the media type offered at a particular point of time, position, used equipment, etc. In these cases, the decision on which code to be used is made dynamically by the Service Manager.

Following are two tables showing examples of Action Groups and Capability Categories, as shown in Table 1 and Table 2.

G1	Node Computation Capability
G2	Communication model
G3	Graphics
G4	I/O interaction
G5	Multimedia

Table 1 Example Action Groups**.**

C1	PowerfulPDA,
C2	Basic PDA
C3	JavaPhone
C4	SmartPhone
C5	Laptop/PC
…	
C10	Bluetooth Communication
C11	WLAN connection
C12	LAN connection
…	
C20	Mouse
C21	TouchScreen
…	
C30	Audio
C31	Video
C32	Text

Table 2 Example Capability Categories**.**

A set of *Selection Rules* and *Mapping Rules* are provided to select manuscripts and map Action Groups to the corresponding Capability Categories based on the current capability and status snapshots. In our implementation, the Rules are represented using the XDD theory [3] – an expressive XML rule-based, knowledge representation - and executed by XET [16,17] engine – a reasoning engine for XDD. An introduction to the XDD theory and an example of a Mapping Rule modelled in XDD is given in the Appendix. Here for easy understanding, we use plain text to describe the rules.

```
R1: IF Action Group Gi belongs to {G1, G3} AND
ServiceRequest.nodeType="PDA" AND node.Processor>=300MHz
    THEN Gi is mapped to C1.
R2: IF Action Group is G2 AND
ServiceRequest.connectionUsed="Bluetooth"
    THEN G2 is mapped to C10.
R3: IF Action Group Gi belongs to {G1, G3} AND
ServiceRequest.nodeType=("Laptop" OR "PC")
    THEN Gi is mapped to C5.
R4: IF Action Group is G4 AND nodeIOdevice="TouchScreen"
    THEN  G4 is mapped to C21.
R5: IF Action Group is G5 AND multimediaSupport="Speaker"
    THEN  G5 is mapped to C30.
```

The mapping from Action Group to Capability Category can be one-to-one or one-to-many depending on the nature of the functionality. Several strategies can be defined for rules. If several rules are executable, the simplest way is to use the first rule and ignore others. On the other hand, complicated strategies, like priority among rules, can be defined to select among the possible rules.

5.3 The Functionality of the Service Manager

As described in Subsection 5.1, the SM is responsible for the selection of manuscripts, based on the Selection Rules, and the computation of the Mapping table, based on the Mapping Rules.

When a role is to be instantiated at a node, an Initial Service request is sent to the SM. This request indicates the service and the role to be realized, as well as further information on the preferred configuration and context information. The SM will select the manuscript based on this information, and compute a Mapping table of the Capability Category for each Action Group used in the manuscript. The selected manuscript and the Mapping table are sent to the SMI to instantiate the role. The SMI downloads a subset of the Action Library, which contains the action codes that correspond to the action types and Capability Category required by the manuscript and Mapping table.

To obtain a proper understanding of the functionality of the SM, consider the following case – referring to the examples of Action Groups and Capability Categories presented in the subsection 5.2. If an action type belongs to G1 (Node Computation Action Group), and the calculated Capability Category for this Action Group is C5 (Laptop/PC), then all the action codes that belong to C5 will be downloaded by the SMI. This means that when the SMI needs to execute an action that belongs to G1 as specified in the manuscript, it will select the corresponding action code for this action type that is designed for C5, fill out all the parameters by the instance data and execute it.

This selection of execution codes can be dynamically adapted to the changing environment, e.g., a role-figure is moved from one node to another, or a user session is changed from a PC to a PDA. In these situations, capabilities offered by the node will be so different that it is preferable to choose different code that is utilizing these capabilities more efficiently. A Function Update request will be issued to the SM, which re-computes the Mapping table so that the SMI can adjust the execution of actions accordingly. In this way, the parameterized behavior pattern specified in manuscripts can be dynamically modified as different codes can be selected based on different capability categories.

An important aspect of the functionality of the SM is its reduced and flexible computation mechanism. By assigning action types to a few Action Groups in the manuscript and action codes to Capability Categories in the Action Library, we calculate the mapping from only a few Action Groups to Capability Categories, thereby, reducing the amount of computation tasks. At the same time, the resulting Mapping table is far shorter than the direct mapping from action types, and thus greatly saves the communication overhead.

6. Example

In this section, an example scenario based on the application TeleSchool is described to demonstrate the ideas described in the previous sections. TeleSchool is a network based learning application, which provides Real-Time-Lectures, Lecture-on-demand and Student-offline-support. The services include distribution of multimedia information and support both wireless and wired communication.

TeleSchool service is defined by a play consisting of three different roles:

- *SchoolServer:* a server role for managing all the clients running TeleSchool and support for all types of services.
- *SchoolRTLServer*: provides functionality specific for real time lectures.
- *SchoolClient*: an application program that provides user access to TeleSchool service. This includes user interface interaction handling (logon and selection of courses and lectures), multimedia session control and realization.

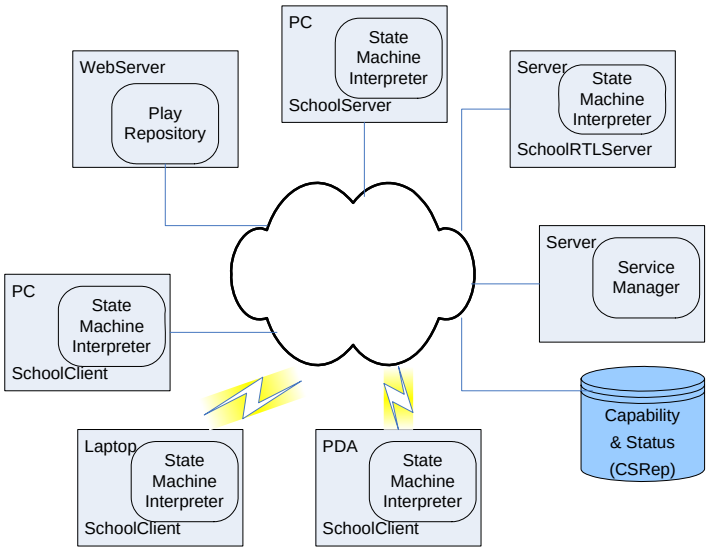


Figure 4 Example TeleSchool Application**.**

An example TeleSchool application is shown in Figure 4. To see how dynamic service management is achieved by the framework proposed in this paper, we look into the specification, selection and adaptation of scenarios based on SchoolClient role. Different versions of manuscripts can be provided for the same role according to the user subscription, e.g., three versions of School Client role - “basic”, “moderate” and “advanced” – can be provided to user.

Service specification defined by the following information should be available before TeleSchool system can provide Real Time Lecture and other services:

Manuscripts for different roles are provided describing parameterized behaviour pattern with assigned *Action Groups*, as examples shown in Figure 5. For the limitation of pages, we only include the most related information in manuscript, i.e., only the actions information is shown. For a more complete example of XML manuscript for the SchoolClient role, please visit the web site: http://tapas.item.ntnu.no/AdapServ. The corresponding action codes are implemented in *Action Library* depending on the

Capability Category. Examples of *Action Groups* and *Capability Categories* are already shown in Table 1 and Table 2 respectively. A set of *Selection Rules* and *Mapping Rules* are provided to select manuscripts and map Action Groups to the corresponding Capability Categories based on the current capability and status snapshots, as described in section 5.

```
<state name="stInit">
 <action>
   <actionType>WindowNew</actionType>
   <param>
     <name>windowSize</name>
     <value>default</value>
   </param>
   <param>
     <name>windowType</name>
     <value>Logon</value>
   </param>
   <actionGroup>G3</actionGroup>
 </action>
 <action>
   <actionType>getUserInput</actionType>
   <storeResult>
     <resultType>msgType</resultType>
     <resultType>msgParameter</resultType>
   </storeResult>
   <actionGroup>G4</actionGroup>
 </action>
 <nextState>stInitUserInterface</nextState>
</state>
<state name="stInitUserInterface">
 <input msg="LogonEventInd">
  <action>
   <actionType>ActorPlugIn</actionType>
   <param>
     <name>role</name>
     <value>SchoolServer</value>
   </param>
   <storeResult>
     <resultType>v_server</resultType>
   </storeResult>
   <actionGroup>G2</actionGroup>
  </action>
  … more actions ….
  <output>
    <msg type="UserVerifyAccessReq">
      <param>
        <name>message</name>
        <value>INPUT_MSG</value>
      </param>
      <dest>v_server</dest>
    </msg>
  </output>
  <nextState>stPasswordIdentify</nextState>
 </input>
</state>
…. More states for user verification and user selection of work type (RealTime or Offline), courses and lectures …
<state name="stSessionCreate">
 <input msg="MMSessionCreate">
  <action>
   <actionType>SessionSetup
   </actionType>
   <param>
     <name>sessionRequired</name>
     <value>INPUT_MSG</value>
   </param>
  </action>
  <actionGroup>G5</actionGroup>
  <nextState>stMediaPlay</nextState>
 </input>
</state>
```

Figure 5 Fragments of example XML manuscript "SC_advanced" for SchoolClient role**.**

Having all the above information available during service specification and design, the TeleSchool service system is ready for deployment and utilization. When the service is deployed and instantiated the first time, SchoolServer and SchoolRTLServer objects will be installed and configured. SchoolClients can be plugged in later on at any possible node running the required TAPAS support. Up-to-date capability and status information is provided by TAPAS system in CSRep, which contains snapshots of the system as the example shown in Figure 6.

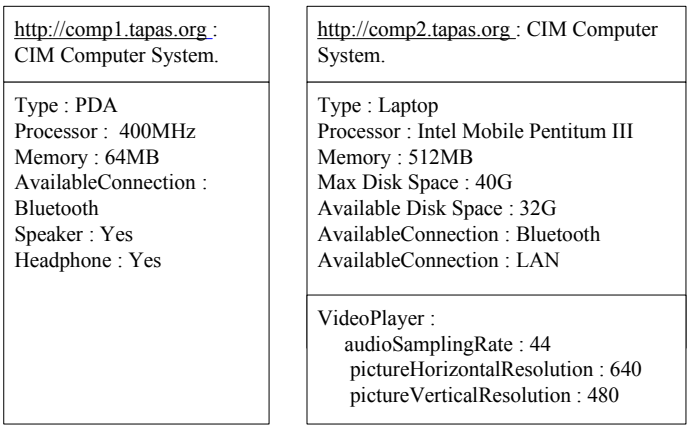


Figure 6 Capability and Status snapshots for two devices: one PDA and one Laptop with Video Player**.**

Our story begins when a student Joe gets on a train and decides to attend some lectures while he travels. He switches on his PDA and asks for TeleSchool services. An Initial Service request as shown in Figure 7 is sent to the SM. The request contains the relevant context information which can not be obtained from CSRep, i.e., the connection context (Bluetooth) and user subscription (advanced). Other context information can be inferred from CSRep, such as the node context (PDA), and location context (mobile). SM decides that the manuscript to be used is SC_advanced according to the serviceType and userSubscription information in the service request. It then computes a Mapping table from the Capability & Status information shown in Figure 6 according to the Mapping Rules R1-R5. The result is shown in Figure 9. The manuscript together with the Mapping table will be sent to the PDA where the SMI will interpret the manuscript and select codes from the subset of Action Library according to the Mapping table. All these codes are optimized to execute on such an environment as Joe's case, e.g., only text information is sent to PDA as it has no graphic display. Joe begins with logon process, selects the courses and lectures he wants to join and follows the lecture as he travels.

After a while, as he becomes interested in a multimedia presentation given in this course, he decides to transfer his session from PDA to his Laptop using wireless connection. As Laptop is equipped with a video player, he can enjoy a multimedia session when his application session is transferred. A Funtion Update Request, as indicated in Figure 8, is sent to the SM, which recomputes the Mapping table, allowing multimedia streaming and better user interface. During his travel, the wireless connection can experience different transmission characteristics, which means his service needs to adapt to this dynamically. For example, if the transmission speed drops dramatically, insufficient capability will result in a new Function Update Request, which will keep only one session and close all other sessions. When the transmission rate increases, some of the closed sessions can be reopened. These will be handled by different Function Update Requests which results in optimal codes selection under the situation. During the whole process, Joe can freely switch to the device he prefers and select the different service capability, and the service system will take care of the optimal configuration and behaviour based on the context and capability information.

```
<InitialServiceRequest>
   <serviceType>TeleSchool</serviceType>
   <roleRequesting>SchoolClient</roleRequesting>
   <preferredConfiguration>
     <nodeInstalling>http://comp1.tapas.org</nodeInstalling>
   </preferredConfiguration>
   <contextInfo>
      <connectionUsed>Bluetooth</connectionUsed>
      <userSubscription>Advanced</userSubscription>
   </contextInfo>
</InitialServiceRequest>
```

Figure 7 Initial Service Request**.**

```
<FunctionUpdateRequest>
    <serviceType>TeleSchool</serviceType>
    <roleRequesting>SchoolClient</roleRequesting>
   <preferredConfiguration>
     <nodeInstalling>http://comp2.tapas.org</nodeInstalling>
   </preferredConfiguration>
    <contextInfo>
     <connectionUsed>WLAN</connectionUsed>
     <userSubscription>Advanced</userSubscription>
     <changedCapability>
        <add>VideoPlayer</add>
     </changedCapability>
  </contextInfo>
</FunctionUpdateRequest>
```

Figure 8 Function Update Request when a user changes from PDA to Laptop and a video player is supplied**.**

```
<CapabilityCategory>
<G2, C10>
<G3, C1>
<G4, C21>
<G5, C30>
</CapabilityCategory>
```

Figure 9 Example Mapping table sent to SMI in response to the Initial Service Request as shown in Figure 7**.**

A prototype reasoning and selection engine based on XDD[3] theory and XET[17] has already been implemented to provide the functionality of the SM and to test the feasibility of the framework proposed in the paper. Due to the limitation of pages, only simple examples are shown here. Detailed information is available on the web: http://tapas.item.ntnu.no/AdapServ.

7. Conclusions

It is very complex and challenging for dynamic service management of adaptable systems, where components comes and goes all the time, and the services will be provided by nodes and devices that experience great variation in their available capabilities. We believe that the solution must be based on a flexible specification which can be adapted by a supporting runtime mechanism. Therefore, the paper has outlined a framework for dynamic service management for network-based adaptable services based on modifiable and parameterised behaviour patterns. Service functionality is classified into Action Groups and Capability Category according to the dependability on capability. Service adaptation based on dynamic code selection is realized by the dynamic mapping from Action Group to Capability Category according to the runtime context and capability information. This approach provides an overall solution that meets the dynamic and flexibility requirements of service management of adaptable systems.

This dynamic service management framework extends the TAPAS core platform by providing the intelligence and flexibility of dynamic service selection and adaptation. A prototype selection engine for SM has been implemented to illustrate the feasibility of the framework proposed in the paper. Work is in progress to integrate this framework with TAPAS system (to integrate SM and the TAPAS core platform) so that an integral platform from service creation to provision and adaptation management can be achieved. More cases need to be studied and applied, e.g., the management of role figure mobility.

References

1 Aagesen, F. A., Helvik, B., Johansen, U. and Meling, H. (2001) Plug and Play for Telecommunication Functionality: Architecture and Demonstration Issues. *Int'l Conf. Information Technology for the New Millennium (ICT'2003)*, Bangkok, Thailand. [http://tapas.item.ntnu.no]

2 Aagesen, F. A., Anutariya, C., Shiaa, M. M. and Helvik, B. E. (2002) Support Specification and Selection in TAPAS. *Proc. IFIP WG6.7 Workshop on Adaptable Networks and Teleservices*, Trondheim Norway, September 2002. [http://tapas.item.ntnu.no]

3 Wuwongse, V., Anutariya, C., Akama, K. and Nantajeewarawat, E. (2001) XML Declarative Description (XDD): A Language for the Semantic Web. *IEEE Intelligent Systems* 16(3): 54–65.

4 Shiaa M. M and Aagesen. F. A. (2002) Architectural Considerations for Personal Mobility in the Wireless Internet, *Proc. IFIP TC/6 Personal Wireless Communications (PWC'2002)*, Singapore, Kluwer Academic Publishers, October 2002. [http://tapas.item.ntnu.no]

5 Jiang, S. and Aagesen, F. A. (2003) XML-based Dynamic Service Behaviour Representation. *NIK'2003*. Oslo, Norway, November 2003. [http://tapas.item.ntnu.no]

6 Inoue, Y., Lapierre, M. and Mossotto, C., *The TINA Book: A Cooperative Solution for a Competitive World*, Prentice Hall, 1999.

7 Tennenhouse D. L., Smith J. M., Sincoskie D., Wetherall D. J. and Minden G. J. (1997) A Survey of Active Network Research. *IEEE Communications*, Vol.35, No1
8 Jeff Kramer and Jeff Magee. The Evolving Philosophers Problem: Dynamic Change Management. *IEEE Transactions on Software Engineering*, 16(11):1293-1306, November 1990.
9 Christine R. Hofmeister and James M.Purtilo. Dynamic reconfiguration in distributed systems: Adapting software modules for replacement. *In Proceedings of the 13th International Conference on Distributed Computing Systems, IEEE Computer Society Press*, May 1993.
10 M. Brunner and R. Stadler, Service Management in Multiparty Active Networks, *IEEE Communications Magazine*, Vol. 38, No. 3, pp. 144-151, March 2000.
11 Massachusetts Institute of Technology, Active Networks, http://www.sds.lcs.mit.edu/activeware/
12 Colombia University, Department of Computer Science, NetScript, http://www.cs.columbia.edu/dcc/netscript/
13 U.S. Department of Defense, Advanced Technology Office, http://www.darpa.mil/ato/programs/ activenetworks/actnet.htm.
14 Raza S.K. and Bieszczad A., "Network Configuration with Plug and Play Components", *Proc. 6th IFIP/IEEE International Symposium on Integrated Network Management*, May, 1999.
15 The IBM autonomic computing project. http://www.research.ibm.com/autonomic/
16 Akama, K., Shimitsu, T. and Miyamoto, E. (1998) Solving Problems by Equivalent Transformation of Declarative Programs. *J. Japanese Society of Artificial Intelligence (JSAI)*, 13(6), 944–952.
17 Anutariya, C., Wuwongse, V. and Wattanapailin, V. (2002) An Equivalent-Transformation-Based XML Rule Language. *Proc. Int'l Workshop Rule Markup Languages for Business Rules in the Semantic Web*, Sardinia, Italy.
18 Parlay. http://www.parlay.org/.
19 JAIN initiative. http://java.sun.com/products/jain.
20 BPEL4WS Specification. http://www-106.ibm.com/developerworks/library/ws-bpel/.

Acknowledgement

The authors would like to thank reviewers for their valuable comments and suggestions and also thank Parami Supadulchai for making the demonstration available on the web.

Appendix

XML Declarative Description (XDD) [3] is an expressive XML rule-based knowledge representation, which extends ordinary, well-formed XML elements by incorporation of variables for an enhancement of expressive power and representation of implicit information into so called *XML expressions*. Ordinary XML elements—XML expressions without variable—are called *ground XML expressions*. Every component of an XML expression can contain variables, which are prefixed by "$***T***:",

where ***T*** denotes the types. For example, **$S:NodeID** is a variable which can be specialized into a string. An *XDD description* is a set of *XML clauses* of the form:

$H \leftarrow B_1, \ldots, B_m, \beta_1, \ldots, \beta_n,$

where $m, n \geq 0$, H (Head) and the B_i (collectively called Body of the clause) are XML expressions, and each of the β_I is a predefined *XML constraint.* The meaning of an XDD expression is the set of all XML elements, which are directly described by and are derivable from its clauses. The head of an XML clause intuitively models the consequence part, while the body describes the antecedence or the conditional part. Thus, each XML clause can be easily interpreted as: deriving the information represented by its head if all the conditions specified in its body hold.

According to this theory, a Mapping Rule is expressed by an XML clause. Its head specifies the mapping from Action Group to Capability Category, and its body describes the requirements and constraints for the capability and status information for the mapping. The Mapping Rule R1 used in section 5 can thus be expressed in XDD formalism as Figure 10.

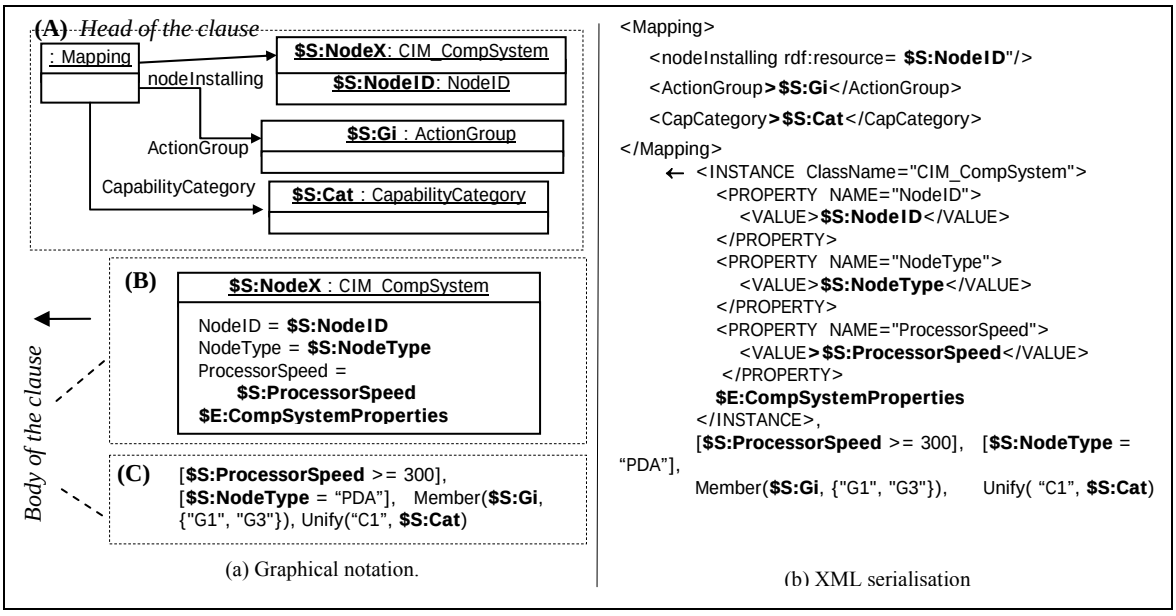


Figure 10 An XDD formalism for Mapping Rule R1**.**

PAPER G: An XML-Based Framework for Dynamic Service Management

Mazen Malek Shiaa, Shanshan Jiang, Paramai Supadulchai and Joan J. Vila-Armenegol

Published in
Proceedings of IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM'04)

Bangkok, Thailand , November 23 - 26, 2004.

Lecture Notes in Computer Science (LNCS) 3283, pp. 273-280, 2004.

An XML-based Framework for Dynamic Service Management

Mazen Malek Shiaa

Shanshan Jiang

Paramai Supadulchai

Joan J. Vila-Armenegol

Abstract Service systems are likely to be highly dynamic in terms of changing resources and configurations. On the one hand, resources are increasingly configurable, extendable, and replaceable. On the other hand, their availability is also varying. For this reason, the handling of these changes is crucial to achieve efficiency. To accomplish this objective, a framework for dynamic service management with respect to service specification and adaptation is proposed.

1. Introduction

A service system, in general, is viewed as a composition of service components. In the lifecycle of service systems (or service components) there are two main phases: the service specification and the service execution phases. The first handles the way services being specified, while the second comprises all the tasks related to service instantiation, operation and maintenance. Historically, service management as a concept has always been discussed and disputed within the second phase only, i.e. independently from the *Service Specification.* Manual modification of the service specification and thereafter configuration and reconfiguration are therefore needed. The concept of *Dynamic Service Management* will take a different approach to service management. It will propose procedures that will make services adaptable to the dynamic changes in their execution environment, based on modifiable and selectable service specifications.

In this paper, we propose a framework for Dynamic Service Management, which addresses two key issues; *Service Specification* and *Service Adaptation*. The main idea is to use behaviour specifications with generalized action types as the service specifications. The actual executing code, or the action libraries that include routines specific to the execution environment, is determined during run-time. The system resources are represented by the so-called *Capability* and *Status* of the system, which characterize all the information related to resources, functions and data inherent to a particular node and may be used by a service component to achieve its functionality. *Service Adaptation* is achieved by allowing the service components to dynamically modify their functionality by requesting changes to their service specification. The framework uses web services to manage the availability and communication of service components.

2. Related Work

Service management, and dynamic service management, has been dealt with in a number of papers, e.g. in active networks [1], replacement of software modules in [2]. In general, the principles discussed in [3] are considered to serve as a basis for any dynamic change management, which are also followed in this paper. Another approach, sharing our view of *Capability* availability, can be found in [4]. It addresses the problem of providing information access to mobile computers using the principle of adjusting the data according to the environment status. Each application is responsible for deciding how to exploit available resources by selecting the fidelity level of the transmitted data. The adaptation is based on choosing between different versions of the data (fidelity levels) in order to match the resource availability. Our work targets the adaptation of any kind of behaviour, instead of the adaptation of data itself. It requires that the behaviour is rich in its processing possibilities. In this regard, the support platform (that executes in the nodes) is responsible for monitoring resource availability, notifying applications of relevant changes to those resources and enforcing resources allocation decisions. Each application is responsible for deciding how best to exploit available resources.

3. Service Specification

As a basic assumption in the framework, a service is viewed as a composition of one or more service components (we also consider a service as a *play* consisting of different *roles*.) Each Service Component is realized or carried out by one or more Role-Figure (being an entity in the architecture that is capable of representing some well-defined functionality). A Role-Figure is realized by a software component (or a collaboration of software components, e.g. multi-threaded processes) executing in a node. However, throughout this paper, and as an abstraction from the implementation domain, Role-Figure will be the constituent of the architecture that is used to provide a basis for service specification and instantiation. In this regard, a Role-Figure specification is the service (or part of a service) specification, which gives a service behaviour description. The most intuitive way to model such a specification is by a State Machine model (one well-known and applied model is the Extended Finite State Machine, EFSM, that is considered here.) [5]. Fig. 1 shows the EFSM data structure for the Role-Figure specification. The behaviour description defined in the Role-Figure specification consists of *states*, *data* and *variables*, *inputs*, *outputs*, and different *actions*. Actions are functions and tasks performed by the Role-Figure during a specific state. They may include calculations on local data, method calls, time measurements, etc. The <*Action*> list in the Role-Figure specifications specifies only the action type, i.e. the method name, and parameters for the action. Each action type belongs to some Action Group according to the nature of action.

Declaring Actions by their general types and classifying them into Actions Groups (e.g. *G1*: Node Computation Capability, *G2*: Communication model, *G3*: Graphics, *G4*: I/O interaction) is a technique used to tackle the problem of platform and implementation independence, as well as achieving a better flexibility and reusability in service and application design. Using Action Types and Action Groups in the service specification keeps the specification short, clean and abstract from how it would

eventually be implemented in different end-user devices, operating policies, and executing environments. For instance, actions such as terminate, exit, error handling, etc. can be classified in "*G100: Control Functions*". Consequently, the action *terminate* would be used in a specification as: <actionType>*terminate*</actionType> <actionGroup>*G100*</actionGroup>

The *Capability* concept abstracts all the information related to functions, resources and data required by the Role-Figures to achieve their functionality. Examples of such capabilities can be: software/hardware components, such as units of processing, storage, communication, system data, etc. Role-Figures achieve tasks by performing or executing actions. Such actions would naturally consume or require resources, or capabilities. A Role-Figure specification explicitly specifies what sorts of capabilities are required. Occasionally, the execution of the Role-Figure halts if certain capabilities are not available. *Status* comprises observable counting measures to reflect the resulting state of the system.

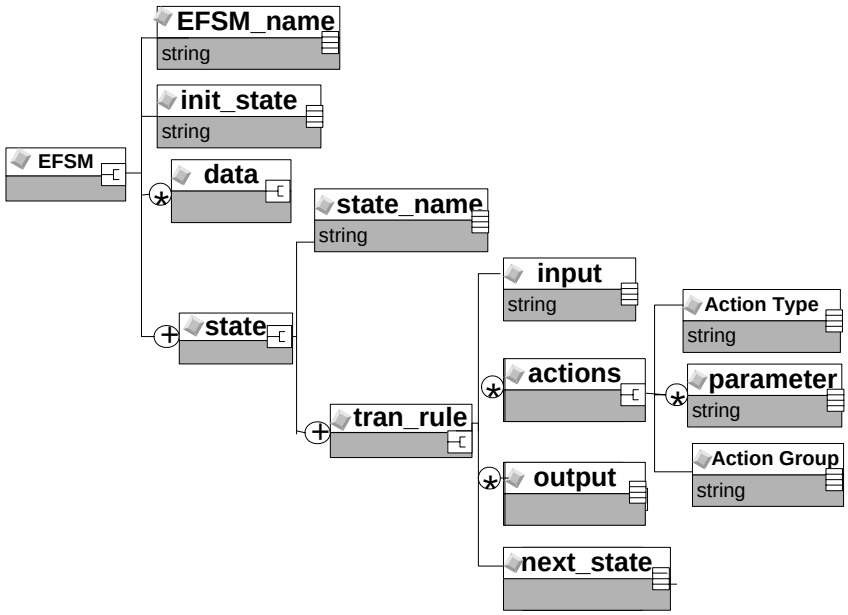


Fig. 1. EFSM model data structure

As have been indicated previously, Action Types, and eventually their classes or Action Groups, are not "*executables"* that may be run in a specific environment or device. Therefore, service designers should map their actions to executable routines provided by device manufacturers (Usually using built-in function calls, with explicit and proper parameter values, through the device's Application Programming Interface, or API.) In this regard, these executable routines should also be classified, to indicate what operating circumstances and capability requirements, they are working within and demanding for. We refer to this classification as *Capability Categories,* so that each category represents a capability set. Examples of Capability Categories can be: *C1*: Powerful PDA, *C2*: Basic PDA, *C10*: Bluetooth, *C11*: WLAN, *C100*: Default CC for G100. A mapping is therefore required to link the action definitions in the service specification to the executable routines stored in Action Libraries.

4. Dynamic Service Management Framework

The Framework presented here considers three distinct forms of Role-Figure specification. Firstly, Role-Figure specification exists as a static representation of the behaviour of the Role-Figure functionality. Secondly, Role-Figure specification would exist as an instantiated code or class instances, with all the necessary mappings to their executing environment, which is "*instantiated Role-Figure specification*." However, a third form may exist between these two forms. Once the capability category is determined, it is important to extend and convert certain actions into corresponding sets of actions, e.g. providing extra security and authentication checks. This form we refer to as "*calculated Role-Figure specification*."

The framework for Dynamic Service Management is illustrated in Fig. 2. The components of the framework are:

1. *Play Repository:* a data base that contains the service definitions and includes:
   - *Role-Figure Specifications*: provide behaviour definitions for each Role-Figure, including the Action Types to be performed and their corresponding Action Groups.
   - *Selection Rules*: provide information for dynamically selecting the proper Role-Figure Specification, if it has several corresponding specifications.
   - *Mapping Rules:* specify the mapping between capabilities and capability categories.
2. *Capability and Status Repository (CSRep):* is a database that provides a snapshot of the resources of the system. It maintains information on all capability and status data in all system nodes.
3. *Action Library:* is a database that contains codes for the state machine-based actions. These codes are implemented according to the capability category they require.
4. *Service Manager (SM):* is responsible for the handling of *Initial Service requests* (to instantiate a Role-Figure), *Role-Figure move* (to move an already instantiated Role-Figure from one node to another), and *Function Update requests* (to change the functionality of an already instantiated Role-Figure). It, first, selects a specific Role-Figure specification based on the *Selection Rules.* Secondly, it calculates the offered Capability Category according to the *Mapping Rules*, which is denoted as *Mapping table.* Then it generates the *calculated Role Figure Specification* by adding the corresponding Capability Category information and the substructures that can be used for decision-making when capability changes occur. This *calculated Role Figure Specification* and the *Mapping table* are then sent to the proper *State Machine Interpreter* for execution.
5. *Requests:* supply, on the one hand, the identification of the Role-Figure to be instantiated or modified. On the other hand, they provide the information to be taken into consideration during the calculation of the Capability Category. Three types of service requests may be handled by the SM:
   - *Initial Service request* indicates a role to be executed in a node.
   - *Role-Figure move* is issued when there is a severe deterioration in certain capabilities availability or the Role-Figure is requested to move to achieve a mobility task for instance. [6] gives an overview of the mobility management of Role-Figures.

- *Function Update request* is issued to update a functionality due to capability change.

6. *Results:* are the outcomes of the calculations performed by SM, which contains the following:
   - *Calculated Role-Figure Specification* indicates a changed Role-Figure specification.
   - *Mapping table* is the result of calculating and matching of Capability Categories based on the given instantaneous capability situation, Mapping rules, and incoming request and its parameters.
7. *State Machine Interpreter (SMI*): is a State Machine execution support [5]. This is the primary entity in the framework responsible for the execution of Role-Figures according to the *instantiated Role-Figure Specification*. The framework allows for a decentralized computation. The dotted-arrow connecting the *Play Repository* and the SMI allows for the calculation of the Role-Figure Specification to be conducted in the node where it will execute, i.e. by the SMI instead of the SM that is, in most cases, would exist at a remote location. This option can solve problems related to over-loaded SM, congested network, time-critical applications, etc.

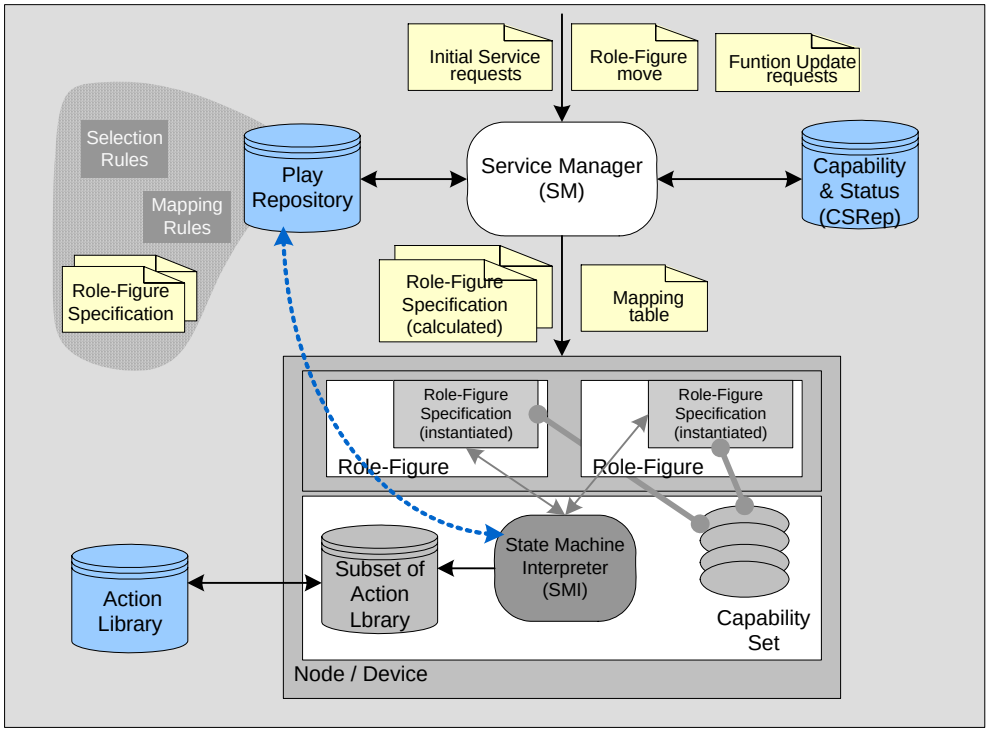


Fig. 2. Dynamic Service Management Framework

The next specification is of a *clientMultimediaPlayer* Role-Figure that runs on a Laptop, with both the capability of WLAN (default) and Bluetooth (used when WLAN is unavailable).

```
<state name="stMediaPlay">
  <actionType>MediaPlay</actionType>
  <actionGroup>G2</actionGroup>
  <CapCategory>C10</CapCategory>
  <CapCategory>C11</CapCategory>
  <Config>
    <defaultCC>C10</defaultCC>
    <ProblemType>
      <List>out of coverage</List>
      <List>congestion</List>
    </ProblemType>
    <choice>
      <check>
        Check Bluetooth neighbourhood
      </check>
      <substate name="Bluetooth">
        <condition value="available">
        <output>
          <msg type="FunctionUpdate">
            <param>
              <name>manuscript</name>
              <value>SchoolClient</value>
            </param>
            <param>
              <name>C10</name>
              <value>out of coverage
              </value>
            </param>
            <param>
              <name>Wireless Communication
              </name>
              <value>unavailable</value>
            </param>
            <dest>ServiceManager</dest>
          </msg>
        </output>
        <nextState>stWaitForManuscript
        </nextState>
      </substate>
      <substate name="NoBluetooth">
        <condition value="unavailable"
                   offline="No">
        <actionType>Terminate</actionType>
        <actionGroup>G100</actionGroup>
        <CapCategory>C100</CapCategory>
        <nextState>stTerminate</nextState>
      </substate>
      <substate name="Offline">
        <condition value="unavailable"
                   offline="Yes">
        <actionType>ChangeToOffline
        </actionType>
        <actionGroup>G100</actionGroup>
        <CapCategory>C100</CapCategory>
        <nextState>stWaitUserInput</nextState>
      </substate>
    </choice>
  </Config>
  <nextState>stWaitUserInput</nextState>
</state>
```

Part of a calculated specification for a *clientMediaPlayer* Role-Figure, in which an action of simple communication has been converted into a structure of additional actions and substates.

5. Implementation issues

The framework has been developed and implemented as part of the TAPAS architecture, see [6,7,8] and the URL: http://tapas.item.ntnu.no/. The implementation of the framework is built around the support functionality of the TAPAS core platform. Java Web Services Developer Pack (Java WSDP) [9] was applied to develop the main communication parts of the framework, while Apache Axis [10] was used as a SOAP server. In this regard, nodes running the platform will have an entity that supports Web Services requests and replies. Fig. 3 shows a possible implementation of the framework, in which a configuration of two nodes running two and three distinct Role-Figures is applied, beside a node running the SM and a web server containing the repository data. The TAPAS Core Platform has been extended with Web-services communication routines, node registry capabilities, and extended configuration data reflecting the reachability of SM. In the figure a connection is highlighted between the CSRep and the nodes participating in the execution of these Role-Figures. Although it has not been fully implemented, a capability registration and update mechanism is considered, which keeps the CSRep updated in terms of any capability change in the nodes. Throughout the experimentation process such update has been conducted manually in order to simplify the overall processing.

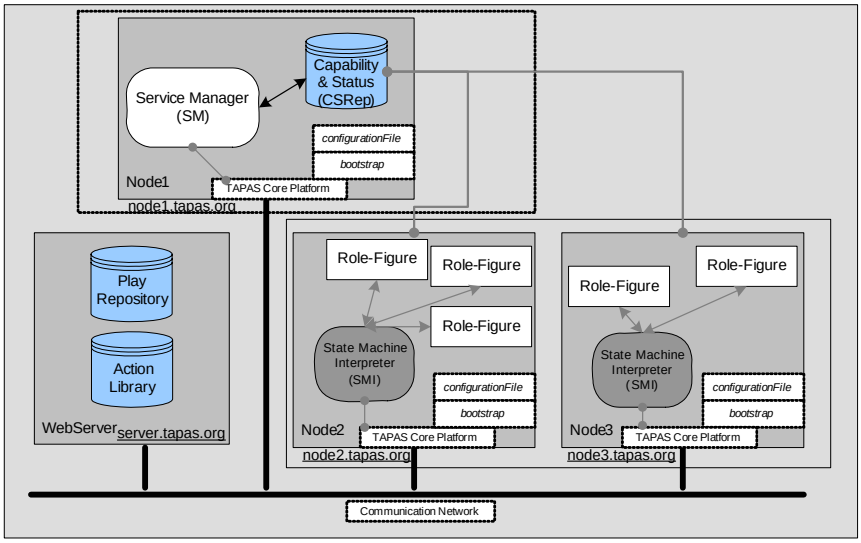


Fig. 3. An implementation of the Dynamic Service Framework within the TAPAS platform

6. Experimentation Scenarios

Several scenarios have been proposed to demonstrate the applicability and foremost features of the framework. During the experimentation, simple application scenarios have been used. The application used was the Teleschool, which is an application facilitating distance-learning, allowing students and teachers to participate in virtual class activities in real-time or off-line modes, using multimedia capabilities on various types of terminals and devices. The test scenarios were limited to run and execute the client Role-Figures, and examine their proper functionality. Here we instantiate *SchoolClient* Role-Figure on a node featuring a Bluetooth and WLAN communications. Below we show an example of a *Role-Figure move* request to initiate a move functionality of a Role-Figure from a node to another one.

```
<RoleFigureMoveRequest>
  <sender>
    <oNode>http://Node2.tapas.org</oNode>
  </sender>
  <dateTime />
  <serviceType>Teleschool</serviceType>
  <roleRequesting>SchoolClient
  </roleRequesting>
  <RFSUsed>SchoolClient_Advanced
  </RFSUsed>
  <preferredConfiguration>
      <nodeInstalling>
          http://Node3.tapas.org
       </nodeInstalling>
  </preferredConfiguration>
  <contextInfo>
      <connectionUsed>LAN</connectionUsed>
      <userSubscription>Advanced
      </userSubscription>
      <MMSupport>VideoPlayer</MMSupport>
  </contextInfo>
  <stateInfo>
      <currentState>stInitUserInterface
      </currentState>
      <variables>
          <variable>
              <name>v_server</name>
              <value>aaaaaa</value>
          </variable>
      </variables>
  </stateInfo>
</RoleFigureMoveRequest>
```

An example *Role-Figure move* request used to move a Role-Figure from Node2 to Node3

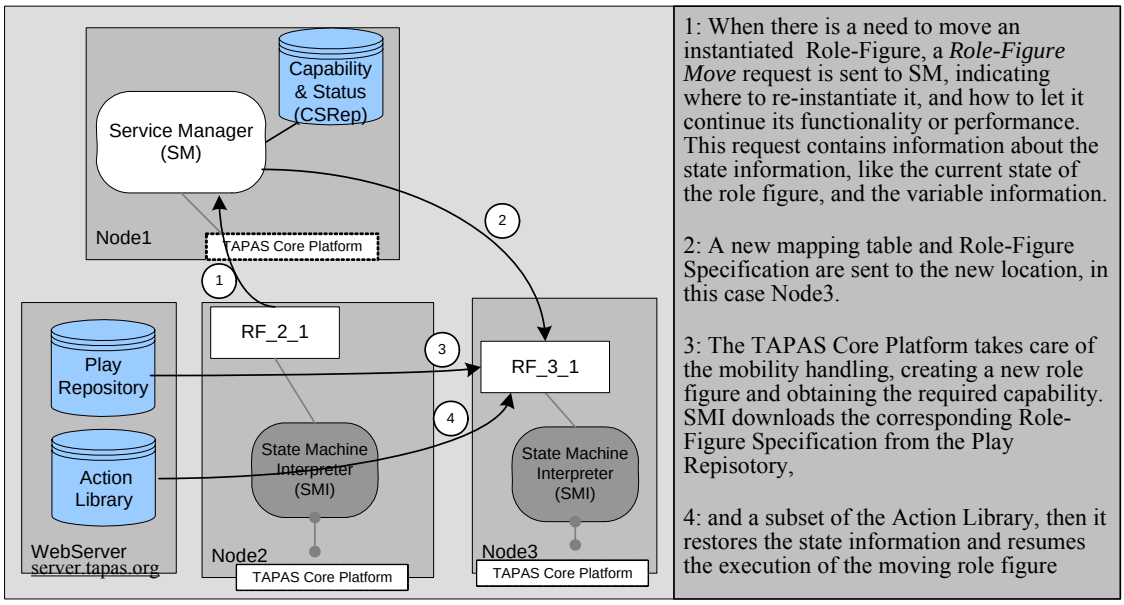


Fig. 4. Experimentation scenario of a Role-Figure move including two nodes

7. Conclusion

In this paper some challenges of the Dynamic Service Management have been discussed, and a framework has been proposed to tackle them. In a highly dynamic system, components composing the system come and go, as well as the system resources that are allocated for them vary and change all the time. The demonstrated framework provides a way of enabling the dynamic service management based on specifications that can be selected, their behavior be computed, and their handling of system resources are all based on the available capabilities in the execution environment. Specifications contain only Action Types and Action Groups, while the executable code or Action Libraries are available on a different database. This distinction is mainly to achieve flexibility. Capability Categories classify these executables based on the capability information in the system.

References

1 M. Brunner and R. Stadler, Service Management in Multiparty Active Networks, *IEEE Communications Magazine*, Vol. 38, No. 3, pp. 144-151, March 2000.

2 Christine R. Hofmeister and James M.Purtilo. Dynamic reconfiguration in distributed systems: Adapting software modules for replacement. *In Proceedings of the 13th International Conference on Distributed Computing Systems, IEEE Computer Society Press,* May 1993.

3 Jeff Kramer and Jeff Magee. The Evolving Philosophers Problem: Dynamic Change Management. *IEEE Transactions on Software Engineering,* 16(11):1293-1306, Nov. 1990.

4 B.D. Noble and M. Satyanarayanan. Experience with adaptive mobile applications in Odyssey (1999). *Mobile Networks and Applications*, 4, 1999.

5 Jiang, S. and Aagesen, F. A. (2003) XML-based Dynamic Service Behaviour Representation. *NIK'2003*. Oslo, Norway, November 2003. [http://tapas.item.ntnu.no]
6 Shiaa M. M and Aagesen. F. A. (2002) Architectural Considerations for Personal Mobility in the Wireless Internet, *Proc. IFIP TC/6 Personal Wireless Communications (PWC'2002)*, Singapore, Kluwer Academic Publishers, October 2002. [http://tapas.item.ntnu.no]
7 Aagesen, F. A., Anutariya, C., Shiaa, M. M. and Helvik, B. E. (2002) Support Specification and Selection in TAPAS. *Proc. IFIP WG6.7 Workshop on Adaptable Networks and Teleservices*, Trondheim Norway, September 2002. [http://tapas.item.ntnu.no]
8 Aagesen, F. A., Helvik, B. E., Anutariya, C., and Shiaa M. M. (2003) On Adaptable Networking, *Proc. 2003 Int'l Conf. Information and Communication Technologies (ICT 2003)*, Thailand.
9 SUN Microsystems, the Web services Homepage, Java Web Services Developer Pack (WSDP) documentation, http://java.sun.com/webservices/index.jsp.
10 The Apache homepage, Apache Axis 1_1, http://ws.apache.org/axis.

PART III: APPENDICES

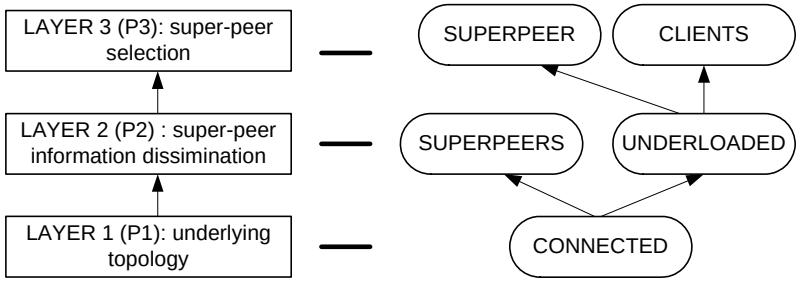
APPENDIX A: Algorithms for Construction and Maintenance of Super-peer SONs

This appendix describes the algorithms for construction and maintenance of super-peer SONs, which is based on a modified version of SG-1 [Mon04]. We first describe the basic idea and algorithms for SG-1 in Section A.1 and Section A.2, then describe our modifications for the construction and maintenance of super-peer based SONs in Section A.3.

The network consists of a large collection of *nodes* (also called *peers*) that are assigned unique identifiers and that communicate with other nodes through message exchanges. The nodes are connected through an *existing routed network* (i.e. the physical underlying network), such as Internet. The super-peer network is constructed as an additional *overlay*, superimposed over the existing connected topology (the *underlying topology*). In the overlay network, each node has a set of *neighbours* that it can communicate directly. Nodes differ in their capabilities, such as the computational power, storage and bandwidth. In order to abstract all such characteristics in a single quantity, the concept *capacity* is used to represent the various capabilities. The *capacity* of node *n* is defined as the maximum number of other nodes it can manage, denoted as c_n. In a super-peer network, each node is assigned as either super-peer role or client role depending on their capacities. A super-peer has higher capacity and manages its clients. The assignment of roles is not permanent: a node may start as super-peer and later change into client of other node with higher capacity.

A.1 The Structure of the Protocol Stack and Datasets in a Node

The algorithm for the construction and maintenance of super-peer network is based on a three-layered protocol stack [Mon04], as shown in Figure A.1(a). All the protocols are gossip-based [DGHI87], where each node periodically exchanges node information with a randomly selected peer. Each node maintains four different datasets, shown in Figure A.1(b), which are maintained by these three protocols respectively.



(a) Protocol stack (b) Datasets maintained in each node

Figure A.1. Three-layered protocol stack run on each node and the datasets maintained in each node.

Layer 1 protocol (P1) maintains the set CONNECTED, which contains the neighbours forming the underlying network. P1 adopted in the experiment is NEWSCAST [JKS03], a gossip-based protocol that maintains an approximately random topology.

Layer 2 protocol (P2) maintains two sets. The set SUPERPEERS contains a random sample of nodes currently acting as super-peers. The set UNDERLOADED contains a subset of SUPERPEERS, comprising super-peers that can accept additional clients, i.e., super-peers that have not reach the maximum number of nodes they can manage. P2 is also a variant of NEWSCAST, which selects random peers from the layer 1 CONNECTED set for information exchange.

Layer 3 protocol (P3) manages the client/super-peer relationship. For a super-peer, CLIENTS set contains the clients managed by each super-peer. For a client, only the connected super-peer is stored. P3 performs the super-peer selection algorithm. It periodically samples the UNDERLOADED set to discover if a client transfer or a role change is required.

A.2. Gossip-based Protocols

All the protocols are gossip-based. Figure A.2 illustrates the gossip paradigm. In particular, P1 and P2 are NEWSCAST protocols, which are used to build and maintain an approximately random topology [JKS03].

Figure A.2. The gossip paradigm. Notation: q is the remote peer; s_q is the state received from q (from [Mon04]).

Each node has a local state, or called *partial view* in NEWSCAST, which consists of a fixed-size set of *peer descriptors*. The size of the partial view is denoted as *s*. A peer descriptor contains node status information such as node identifier, capacity, current role, and timestamp. Each node executes two threads. At periodic intervals, the active thread initiates a *state exchange* with a randomly selected neighbour through method RANDOMPEER by sending its partial view though method SENDSTATE and waits for a response from the selected peer. The passive thread waits for a message sent by an initiator and replies with its partial view through method RETURNSTATE. Method UPDATE updates its local state by merging the received partial view with the current local partial view, and keeping the *s* freshest descriptors. In this way, new information enters the system when a node sends its partial view by inserting its own newly created descriptor. At the same time, old information is gradually and automatically removed from the system and is replaced by new information. The gossip intervals of length δ are called *rounds*. In each round, each node initiates *exactly one* state exchange.

Figure A.3 illustrates an example NEWSCAST exchange between node A (initiator) and node B. Each node has a partial view with 6 descriptors (depicted inside the ellipses). A descriptor shown in the figure is a *<node-ID, timestamp>* pair. After the state exchange, each node merges the received partial view with its own partial view. The result is depicted under the empty arrow: each node selects 6 freshest descriptors at random (as depicted inside the ellipses) and discarded the others.

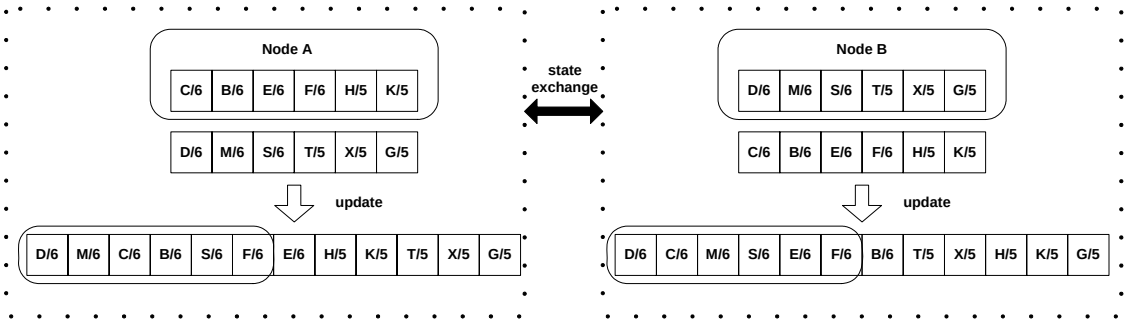


Figure A.3. An example NEWSCAST exchange between node A (initiator) and node B

The generated topology by NEWSCAST is very close to a random graph with out-degree *s*. According to experimental results, choosing s = 20 is sufficient for stable and robust connectivity [JKS03]. In our experiments, we select s = 30.

Although P1 and P2 are both NEWSCAST protocols, they have difference in method implementation. In P1, a node selects a random peer from its current partial view; while in P2, a node selects a random peer from the CONNECTED set managed by P1. In addition, in order to generate the SUPERPEERS and UNDERLOADED sets, the partial view exchanged in P2 contains only descriptors that belong to super-peers or underloaded super-peers, respectively. Since nodes are selected from the CONNECTED set in P2, the information about super-peers is disseminated among all nodes, not only the super-peers. The benefit of this choice is that when super-peers fail, the affected clients are still able to disseminate information about the newly selected super-peers.

P3 is a gossip-based super-peer selection protocol. Its main algorithm is illustrated in Figure A.4. This algorithm is executed only by super-peers. In addition, super-peers and clients periodically probe their counterparts in order to detect failure. If a client fails, it is simply removed form the CLIENTS set. If a super-peer fails, each of its clients needs to probe some nodes in UNDERLOADED set to check if they are willing to accept it as client; if not, it becomes a super-peer and starts to run the super-peer selection algorithm. This function is illustrated in Figure A.5.

```
RANDOMPEER():
   S ← { r | c_r ≥ c_p ∧ r ∈ UNDERLOADED }      %% Select those super-peers that have capacity
   q ← null                                      %% higher than or equal to the local peer capacity.
   while ( S ≠ ∅ ∧ q = null )
        r ← ⟨pick a random node from S⟩           %% UNDERLOADED may contain obsolete info,
        S = S – {r}                               %% multiple selections are made until a node is
        l_r ← ⟨request load from r⟩               %% found that is up and has capacity to accept
        if (l_r < c_r ∧ (c_p < c_r ∨ l_r > l_p)   //Found   %% more clients. If same capacity, the one with
            q ← r                                 %% more clients than the local node will be
   return q                                       %% selected.
```

```
SENDSTATE(q):                                  %% Select as many of the local clients as the
    C ← ⟨select min(c_q – l_q, l_p) local clients⟩   %% chosen peer is able to accept, i.e. the number
    send C to q                                %% of clients to be transferred to the selected peer.
RETURNSTATE(q): send ∅ to q                    %% All local clients have been transferred.
                                               %% In addition, the local peer changes role to
                                               %% client.

UPDATE(C, q):                                  %% This is for the selected peer that has higher
    CLIENTS ← CLIENTS ∪ C                      %% capacity and will accept clients transferred.
    if (l_q == 0 ∧ l_p < c_p)                  %% When all the clients of q has been transferred,
        CLIENTS ← CLIENTS ∪ {q}                %% and the selected peer has more capacity, q will
        ⟨q becomes a client⟩                   %% become a client of the selected peer.
    else if (∃r ∈ CLIENTS: c_r > c_q)          %% Otherwise, if the selected peer has a local
        ⟨transfer clients of q to r⟩           %% client r that has more capacity than q, the
        CLIENTS ← CLIENTS ∪{q} – {r}           %% remaining clients of q will be transferred to r
        ⟨q becomes a client, r becomes a super-peer⟩   %% with role change between r and q.
```

Figure A.4. The super-peer selection algorithm. Notations: p is the local peer; q and r are remote peers; l_q denotes the number of clients of the remote peer q, obtained by an explicit request; l_p denotes the local number of clients (adapted from [Mon04]).

```
DOCLIENT(p):
    ⟨probe p's super-peer⟩                     %% Check the status of p's super-peer.
    if ⟨super-peer fails⟩                      %% If super-peer fails, select a random super-peer
        S ← { r | r ∈ UNDERLOADED }            %% from UNDERLOADED.
        found ← false
        while (!found ∧ S ≠ ∅)                 %% Since UNDERLOADED may contain obsolete
            r ← ⟨pick a random node from S⟩     %% info, multiple choices are made to find a
            l_r ← ⟨request load from r⟩         %% super-peer r that is up and is able to accept
            if (l_r < c_r)   //Found           %% more clients.
                ⟨p becomes client of r⟩         %% If found, p becomes a client of r.
                found ← true
        if (!found)                            %% Otherwise, p becomes a super-peer itself.
            ⟨p becomes a super-peer⟩
```

Figure A.5. Periodic check of super-peer status in a client.

A.3. Construction and Maintenance of Super-peer SONs

In a super-peer SON-based service discovery system, each node is associated with one or several SONs. Each SON has a super-peer, and other nodes in the SON are assigned client role. The above described SG-1 algorithm thus needs to be modified in order to construct super-peer based SONs. Figure A.6 shows the new data model of a node. Each node maintains the following data:

- *NodeId*: the node identifier.
- *Capacity*: the node capacity.
- *ConnectedSet:* the neighbours that the node can communicate with.

- *ServiceDescriptionSet:* the set of service descriptions contained in the node.
- *CategorySet*: the set of service categories corresponding to the SONs that the node joins.
- *SONinfo*: contains SON-related information for each SON that the node joins.
    - *ServiceCategory*: the service category corresponding to the SON.
    - *Role*: the role of this node in this SON, which is either super-peer or client.
    - *SuperpeerInfo*: for each super-peer role, two datasets are defined:
        - *SuperpeerSet*: contains a random sample of super-peers that this node can communicate with.
        - *UnderloadedSet*: the super-peers from the *SuperpeerSet* that can accept more clients.
        - *ClientSet*: contains the clients that this node manages.
    - *ClientInfo*: for a client role, it refers to the super-peer that this node is connected with.

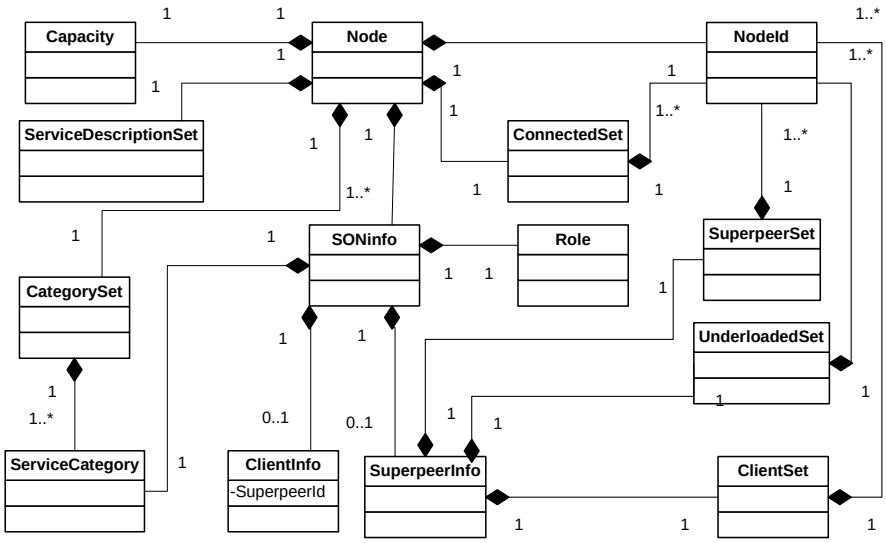


Figure A.6. The data model for a node.

The algorithm for construction and maintenance of super-peer networks needs to be modified accordingly. The main changes are:

- The partial view information exchanged by P1 should include *CategorySet* information. As a result, CONNECTED set contains neighbours for each SON that the local node joins. In other words, the neighbours are associated with service category.
- P2 and P3 are executed for each service category in the *CategorySet* (i.e. for each SON that the node joins).
- At start up, each node is assigned super-peer role. A node can be super-peers for several SONs, but the total number of clients it manages must be smaller than its capacity. As to be noted, this choice may not be the optimal from the resource utilization point of view. We have only experimented with single service

category case, i.e., each node joins only one SON (i.e. the one corresponding to a service category group that has the highest number of service descriptions in a node). The optimal capacity allocation is a challenge in the implementation of multiple service categories case.

APPENDIX B: PeerSim Simulator

The PeerSim simulator [Pee07] is applied for the simulation of constructing and maintaining super-peer SONs. This appendix first gives a brief introduction of PeerSim, and then presents an example simulation of the construction of super-peer SON networks using PeerSim.

PeerSim is an open source, Java based, P2P simulation framework aimed at developing and testing P2P algorithms in a dynamic environment. The main goals for PeerSim are high scalability and support for dynamicity. It is targeted for large-scale environment, and can scale up to 1 million peers. All the algorithms developed in the BISON project [BIS07] have been developed and tested on PeerSim.

PeerSim supports two simulation models:

- *Round-based* (or *cycle-based*): the simulation runs in a sequential order, and in each round, each node runs its protocols sequentially.
- *Event-based*: a set of events (messages) are scheduled and the node protocols are run according to the message delivery time order.

Round-based simulation model is used in our experiment. This model achieves high scalability and performance at the cost of some loss of realism. The simplifying assumptions of round-based model are as follows:

- *Lack of transport layer simulation*. Each node can communicate with each other directly using method calls. The overhead introduced by the low level communication protocol stack such as TCP or UDP is ignored.
- *Lack of concurrency*. Each node is given control periodically, and in a sequential order, when it can perform arbitrary actions (i.e. control or protocol actions).

The main objects in PeerSim simulator are:

- *Node*. Each node object can run many protocols.
- *Protocol.* It defines the operations to be performed at each round. Each protocol uses only local information.
- *Control*. It defines operations that require global network knowledge. Each control object is associated with a *scheduler*, which determines *when* the actions defined in the control object should be performed. Control objects can be further distinguished as *Initializers, Dynamics and Observers*.
    - *Initializer* objects are used to initialize the protocol status and node properties (e.g., node capacity). They are scheduled to run only at the beginning of the simulation.
    - *Dynamic* objects are used to add dynamics to the system behaviour by changing some parameters (e.g. remove or inject nodes at predefined time intervals).
    - *Observer* objects are used to monitor the system properties and gather the system statistics. Both Dynamics and Observers can be scheduled to run periodically or at certain points during the simulation.

PeerSim has a configuration file which is loaded at the beginning of the simulation. The configuration file defines the simulation parameters (e.g. network size, node capacity), the protocols defined for each node, the Initializers, as well as the Dynamics and Observers that need to be run at specified points.

Figure B.1 depicts the execution of the PeerSim simulator.

```
Simulator():
    ⟨load configuration file⟩
    ⟨run initializers⟩
    while (time < MaxRounds ∧ !StopCondition)
        for (each node in the network)
            ⟨run each protocol⟩
        for (each Dynamic and Observer object o)
            if (o.scheduler.active())
                ⟨perform o's actions⟩
```

Figure B.1. The execution of the PeerSim round-based simulator.

Figure B.2 illustrates an example simulation of the construction of super-peer networks. All the 7 nodes want to join the same SON (i.e. have the same service category). Each node has node Id and capacity, depicted in the small circle. The size of the partial view is 3. Each node has three datasets, the CONNECTED set (C), SUPERPEERS set (S) and UNDERLOADED set (U) (cf. APPENDIX A). The dashed lines with arrow constitute the underlying topology. A directed line shows the relationship from a client to its super-peer. At start up, each node assumes super-peer role, shown in Figure B.2(a). In the PeerSim round-based simulation, for each round, each node will sequentially execute all its protocols, namely, P1, P2, and P3. We assume that a node selects the same peer to exchange views for all the three protocols at each round. In real simulation, however, a node will randomly select one peer to exchange views for each protocol. Hence, different protocols may select different peers to exchange.

- When node A starts its super-peer selection algorithm (P3), it first looks for a peer from UNDERLOADED set which has higher capacity than itself. It selects node B, and exchanges state with B. Since node B has higher capacity than node A and is underloaded, node A changes into a client of node B, as shown in Figure B.2(b).
- For node B, it selects node G from its UNDERLOADED set. Since node G has higher capacity than node B and has available capacity to accept both node B and its clients (i.e. node A), node A and B changes into clients of node G. Such role changing and client transfer is shown in Figure B.2(c).
- Similarly, node C exchange with node G and become client of G, as shown in Figure B.2(d).
- Since G has now reached its capacity (i.e. it has 3 clients) and can not accept more clients, node D can only select node E and changes role into client of node E, as shown in Figure B.2(e).

- Node E finds that node F has higher capacity and can further accept node E with its client, therefore, role changing and client transfer occurs as shown in Figure B.2(f).
- For node F, there is no underloaded super-peer that has higher capacity, so it does nothing.
- For node G, node F has higher capacity than it, but node F can only accept 2 more clients. Therefore, node A and node B are transferred as clients of node F. A final check is whether node F has a client with higher capacity than node G. Since no other node has higher capacity than node G, node G remains a super-peer with only one client node C. Therefore, at the end of this simulation round, two super-peers (node F and node G) are selected for the network of 7 nodes, as shown in Figure B.2(g).
- If a super-peer fails (e.g. node G), all its clients need to find new super-peers. Since node F has now reached its maximum capacity, node C decides to be super-peer itself, and waits to participate in another round of super-peer selection. The result is illustrated in Figure B.2(h).

As to be noted:

- Figure B.2(b) – B.2(g) represent one simulation round execution sequence, while Figure B.2(h) shows the result of another simulation round.
- Since partial views are updated at each round by NEWSCAST, the topology generated is in a continuous state of flux. Even when the physical underlying network is stable, the list of neighbours contained in the CONNECTED dataset (representing continuously fresh samples of the network) may still differ after a state exchange. The CONNECTED dataset actually contains the neighbours that the node can potentially communicate via a routed path in the physical underlying network.

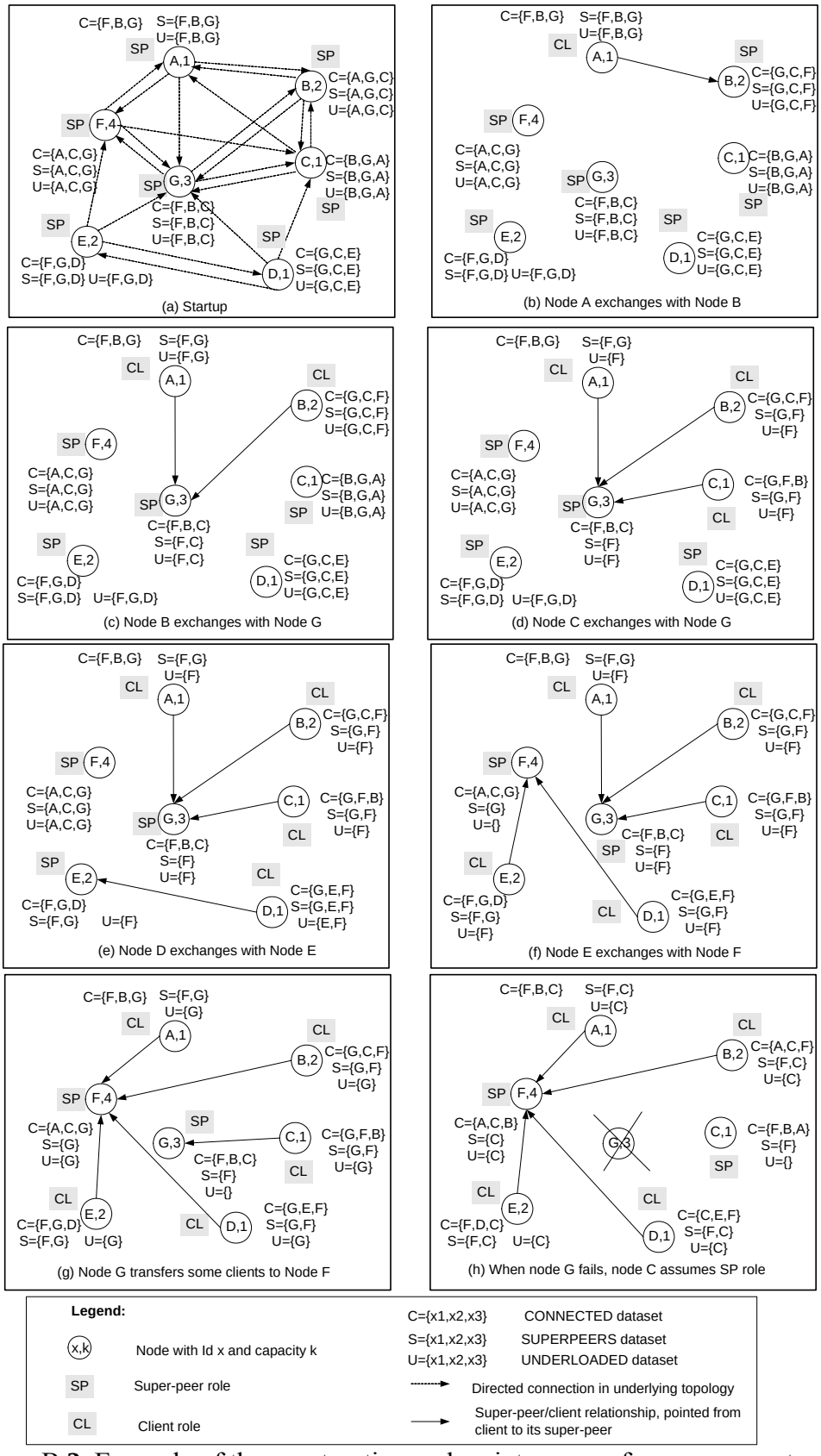


Figure B**.2**. Example of the construction and maintenance of super-peer networks.

APPENDIX C: Additional Simulation Results

This appendix provides additional results from the simulations on super-peer SON service discovery system.

C.1 Discovery Overhead Factor

To normalize messages-per-request, an evaluation measure called *discovery overhead factor* is defined, which is the value of messages-per-request divided by the number of matches (i.e. the retrieved matching services). A small discovery overhead factor means a small messages-per-request value per match. The simulation parameters are the same as those used in the simulations for the evaluation of service discovery efficiency of PAPER C. Figure C.1 shows the average discovery overhead factor for different network sizes when recall=1.0. It shows that SON power-law system has a very small and almost stable discovery overhead factor (0.04-0.06). SON uniform system has a small discovery overhead factor, but it increases with network size. Gnutella system has a high discovery overhead factor for both uniform and power-law distributions and the value also increases with network size.

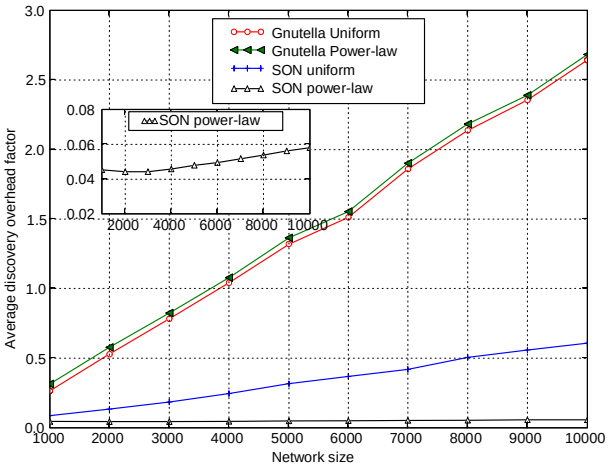


Figure C.1. Average discovery overhead factor for different network sizes when recall=1.0.

C.2 Observed Standard Deviation in Experiments

The results of average messages-per-request and discovery overhead factor shown in Figure 5 in PAPER C and Figure C.1 in this appendix have been averaged over 10 replications, and for each replication, the value is averaged over 25000 service requests. Table C.1, Table C.2 and Table C.3 show the corresponding standard deviation in these experiments. The *mean value* refers to the average of the data, which are plotted in the figures. The mean value along with the standard deviation reflects the dispersion of the data. A high value of standard deviation for a small mean value means that the data are dispersed significantly whereas a low standard deviation for a high value of mean value means the opposite.

Table C.1. The mean value and the standard deviation of messages-per-request value for different network sizes (recall=1.0) for Figure 5 of PAPER C.

Number of Nodes	SON uniform distribution		SON power-law distribution	
	Mean value	Standard deviation	Mean value	Standard deviation
1000	861,7	7,16	383,5	2,06
2000	1358,7	14,43	403,5	3,38
3000	1863,8	16,43	425,7	3,3
4000	2679,9	20,37	445,4	3,89
5000	3182,2	22,58	465,2	4,16
6000	3686,7	22,22	486	2,15
7000	4190,4	22,84	507,7	7,36
8000	5096,8	17,82	525,9	6,63
9000	5600,1	21,39	550,1	4,86
10000	6107,3	24,05	566,2	4,74

Table C.2. The mean value and the standard deviation of discovery overhead factor value for different network sizes (recall=1.0) for Figure C.1 in this appendix (for SON system).

Number of nodes	SON uniform distribution		SON power-law distribution	
	Mean value	Standard deviation	Mean value	Standard deviation
1000	0,1302	0,0249	0,05557	0,01037
2000	0,1462	0,0086	0,05095	0,01853
3000	0,1921	0,0051	0,04515	0,00101
4000	0,271	0,0025	0,04618	0,0006
5000	0,3209	0,0022	0,04784	0,00038
6000	0,3698	0,0022	0,04963	0,00033
7000	0,423	0,0033	0,05181	0,00078
8000	0,5204	0,0278	0,0533	0,00067
9000	0,5632	0,0025	0,05579	0,00054
10000	0,6125	0,0019	0,05728	0,00046

Table C.3. The mean value and the standard deviation of discovery overhead factor value for different network sizes (recall=1.0) for Figure C.1 in this appendix (for Gnutella system).

Number of nodes	Gnutella uniform distribution		Gnutella power-law distribution	
	Mean value	Standard deviation	Mean value	Standard deviation
1000	0,26559	0,01779	0,31545	0,168006
2000	0,52926	0,02364	0,57863	0,18954
3000	0,78345	0,03082	0,82605	0,21174
4000	1,04256	0,03333	1,0793	0,21009
5000	1,3209	0,03799	1,3662	0,26115
6000	1,5127	0,037203	1,5539	0,26899
7000	1,8596	0,04129	1,90105	0,29449
8000	2,1382	0,03975	2,1832	0,32383
9000	2,3555	0,03138	2,3904	0,29958
10000	2,6428	0,03952	2,6819	0,35489

Bibliography

[Aag07] F. A. Aagesen. Lectures on "Network and Service Management". 2007.

[AASH02] F. A. Aagesen, C. Anutariya, M. M. Shiaa and B. E. Helvik. Support Specification and Selection in TAPAS. Proceedings of the IFIP WG6.7 Workshop and EUNICE Summer School on Adaptable Networks and Teleservices, Trondheim, Norway, September 2002.

[AC07] IBM Autonomic computing. http://www.research.ibm.com/autonomic/. Accessed December 2007.

[AHAS03] F. A. Aagesen, B. E. Helvik, C. Anutariya and M. M. Shiaa. On Adaptable Networking. The First International Conference on Information and Communication Technologies, ICT'2003, Assumption University Thailand, April 2003 .

[ANA07] ANA: Autonomic Network Architecture project. http://www.ana-project.org/. Accessed December 2007.

[AS04] S. Androutsellis-Theotokis and D. Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. ACM Computing Surveys, Vol. 36, No. 4, December 2004, pp. 335-371.

[AS07] F. A. Aagesen and P. Supadulehai. A Capability-based Service Framework for Adaptable Service Systems. In Proceedings of The 2nd International Conference on Advances in Information Technology (IAIT2007), Bangkok, Thailand, November 2007.

[ASAS05] F. A. Aagesen, P. Supadulchai, C. Anutariya and M. M. Shiaa. Configuration Management for an Adaptable Service System. IFIP International Conference on Metropolitan Area Networks, Architecture, Protocols, Control, and Management, April 11-13, 2005, Ho Chi Minh City, Viet Nam.

[AWW02] C. Anutariya, V. Wuwongse and V. Wattanapailin. An Equivalent-Transformation-based XML Rule Language. *Proceedings of the Proc. International Workshop on Rule Markup Languages for Business Rules in the Semantic Web, Sardinia, Italy.* June 2002.

[Baj06] S. Bajaj et al. Web Services Policy Framework (WS-Policy). 2006. http://www-128.ibm.com/developerworks/library/specification/ws-polfram/.

[BDD99] H. Berndt, E. Darmois, F. Dupuy, et al. The TINA Book: a co-operative solution for a competitive world. Published by Prentice Hall Europe, 1999.

[Bion07] Bionets project. http://www.bionets.org/. Accessed December 2007.

[BIS07] The BISON Project. http://www.cs.unibo.it/bison. Accessed December 2007.

[Bri91] E. Brinksma. What is the method of formal methods? Forte'91. Sidney, Australia, November, 1991.

[CB06] H. N. Castejon and R. Bræk. Formalizing Collaboration Goal Sequences for Service Choreography. In Proceedings of FORTE'06, September 26-29, 2006, Paris, France. Springer-Verlag, LNCS 4229, 2006.

[CDK99] A. T. Campbell, H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente, and D. Villela. A Survey of Programmable Networks. ACM SIGCOMM Computer Communication Review, Vol. 29, No. 2, pp. 7-23. Apr. 1999.

[CFN04] C. Carrez, A. Fantechi, and E. Najm. Assembling Components with Behavioural Contracts. Annals of Telecomms, 2004.

[CG02] A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. *Technical Report*, Computer Science Department, Stanford University, October 2002.

[DDF06] S. Dobson, S. Denazis, A. Fern´andez, D. Ga¨ıti, etc. A Survey of Autonomic Communications. ACM Transactions on Autonomous and Adaptive Systems, Vol. 1, No.2, pp. 223–259, December 2006.

[DFM00] R. Dingledine, M. Freedman, and D. Molnar. The FreeHaven Project: Distributed Anonymous Storage Service. In Worshop on Design Issues in Anonymity and Unobservability. 67-95. 2000.

[DGHI87] A. Demers, D. Greene, C. Hauser, W. Irish, et al. Epidemic Algorithms for Replicated Database Management. In Proc. of 6th ACM Symp. On Principles of Distributed Computing. PODC'87, Vancouver, August 1987.

[DZ83] J. D. Day, and H. Zimmermann. The OSI Reference Model. Proc. Of the IEEE, vol. 71, pp. 1334-1340. 1983.

[ECAC07] European Commission Autonomic Communication. http://www.autonomic-communication.org/. Accessed December 2007.

[Flo03] J. Floch. Towards Plug-and-Play Services: Design and Validation using Roles. PhD thesis, 2003:47, NTNU, Trondheim, February, 2003.

[Gon01] L. Gong. JXTA: A Network Programming Environment. Internet Computing. IEEE, vol. 5(3), May-June 2001.

[HS04] P. Haase and R. Siebes. Peer Selection in Peer to Peer networks with semantic topologies. In Proceedings of the 13th International World Wide Web Conference (WWW), New York, NY USA. May, 2004.

[IN92] CCITT Recommendation Q.1200: Q Series Intelligent Network Recommendation Structure. March 1992.

[JA03] S. Jiang and F. A. Aagesen. Design and Implementation for XML-based Dynamic Service Behaviour Representation. Plug-and-play technical report, 2. Trondheim: Department of Telematics, NTNU, 2003. ISSN 1500-3868.

[JKS03] M. Jelasity, W. Kowalczyk and M. van Steen. Newscast Computing. Technical Report IR-CS-006, Vrije Universiteit Amsterdam, Department of Computer Science, November 2003.

[Kaz07] Kazaa homepage. http://www.kazaa.com/. Accessed December 2007.

[KM90] J. Krameer and J. Magee. The Evolving Philosophers Problem: Dynamic Transactions on Software Engineering, 16(11):1293-1306, November 1990.

[LW00] Lime Wire LLC. Gnutella – Limewire 4.0. http://www.limewire.com/. 2000.

[Mon04] A. Montresor. A Robust Protocol for Building Superpeer Overlay Topologies. Proceedings of the Fourth International Conference on Peer-to-Peer Computing (P2P 2004), August 25-27, 2004, pp. 202-209.

[MS95] S. T. March and G. F. Smith. Design and Natural Science Research on Information Technology. Decision Support Systems. 15(4): 251-266. 1995.

[NGN04] ITU-T. Recommendation Y.2001. General Overview of NGN. December. 2004.

[NGN07] ITU-T. CSR 2007 Discussion Paper. NGN Overview. Available: http://www.itu.int/ITU-D/treg/Events/Seminars/GSR/GSR07/discussion_papers/Cohen_NGN_Overview_Final.pdf. Accessed December 2007.

[ODP95] ISO/IEC 10746-1; ITU-T X.901. Basic Reference Model of Open Distributed Processing – Part1: Overview. ISO, 1995.

[OWLS03] The OWL Services Coalition. OWL-S: Semantic Markup for Web Services. 2003. http://www.daml.org/services/owl-s/1.0/owl-s.html.

[Pee07] PeerSim: a peer-to-peer simulator. http://peersim.sourceforge.net/. Accessed December 2007.

[PSNS03] M. Paolucci, K. Sycara, T. Nishimura, and N. Srinivasan. Using DAML-S for P2P Discovery. In Proceedings of *ICWS'03*.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems. In *Middleware*, 2001.

[RFHK01] S. Ratnasamy, P. Francis, M. Handley, and R. Karp. A Scalable Content-Addressable Network. SIGCOMM 2001.

[RPOS05] S. Ryu, S. K. Park, D. Oh, G. Sihn, K. Han, and S. Hwang. Research Activities on Next-Generation Mobile Communications and Services in Korea. IEEE Communications Magazine. September 2005.

[SB04] R. T. Sanders and R. Bræk. Modelling Peer-to-Peer Service Goals in UML. In Proceedings of 2nd IEEE Intl. Conf. on Software Engineering and Formal Methods, 2004.

[SBBA05] R. T. Sanders, R. Bræk, G. Von Bochman and D. Amyot. Service Discovery and Component Reuse with Semantic Interfaces. In Proceedings of 12th Intl. SDL Forum, LNCS 3530, pages 85-102, 2005.

[SBF98] R. Studer, V. R. Benjamins and D. Fensel. Knowledge Engineering: Principles and Methods. Data and Knowledge Engineering, 25(1-2):161-197, 1998.

[SCKB05] R. T. Sanders, H. N. Castejon, F. Kraemer and R. Bræk. Using UML 2.0 Collaboration for Compositional Service Engineering. In Proceedings of the 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML 2005). Montego Bay, Jamaica, October 2-7 2005. Springer Verlag, LNCS 3713, 460-475, 2005.

[SDL00] ITU-T. Recommendation Z.100. Specification and Description Language (SDL). Geneva, 2000 (11/99).

[SMK01] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. SIGCOMM 2001.

[Sup07] P. Supadulchai. NxET Reasoning Engine. Plug-and-play Technical Report, Department of Telematics, NTNU, ISSN 1500-3868, 2007.

[SW07] W3C Semantic Web Activity Homepage. http://www.w3.org/2001/sw/. Accessed December 2007.

[SWM04] M. K. Smith, C. Welty and D. L. McGuinness. OWL Web Ontology Language Guide. W3C Recommendation. February 2004.

[TSS97] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, Vol. 35, No. 1, pp80-86. January 1997.

[UML05] Object Management Group. Unified Modelling Language: Superstructure Specification, version 2.0. Needham, USA, August, 2005.

[UML07] UML Reseouce Page. http://www.uml.org/. Accessed December 2007.

[VK04] V. Varishnavi and B. Kuechler. Design Research in Information Systems. Association for Information Systems, January 2004. Available online: http://www.isworld.org/Researchdesign/drisISworld.htm.

[WAAN01] V. Wuwongse, C. Anutariya, K. Akama, and E. Nantajeewarawat. XML Declarative Description: A Language for the Semantic Web. *IEEE Intelligent Systems,* Vol. 16, No. 3, May/June 2001, pp. 54-65.

[WS07] W3C Web Services Activity Homepage. http://www.w3.org/2002/ws/. Accessed December 2007.

[WSDL01] W3C. Web Services Description Language (WSDL) 1.1. March 2001. http://www.w3.org/TR/wsdl.

[XML06] W3C: eXtensible Markup Language (XML) 1.0 (Fourth Edition). August 2006. http://www.w3.org/TR/2006/REC-xml-20060816/.

[YG03] B. Yang and H. Garcia-Molina. Designing a Super-peer Network. IEEE International Conference on Data Engineering (ICDE'03), 2003.

[ZKJ01] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report, UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley. April 2001.