Paramai Supadulchai

Reasoning-based Capability Configuration Management in Adaptable Service Systems

Thesis for the degree philosophiae doctor

Trondheim, January 2008

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

NTNU

NTNU
Norwegian University of Science and Technology

Thesis for the degree philosophiae doctor

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

ISBN 978-82-471-6020-6 (printed version)
ISBN 978-82-471-6034-3 (electronic version)
ISSN 1503-8181

Doctoral theses at NTNU, 2008:10

Printed by NTNU-trykk

Abstract

Networked Service Systems are considered. Services are realized by service components, which by their inter-working, provide a service in the role of a service provider to service users. During more than two decades, *networked service* systems have been an important research topic. Focus was on efficiency in the definition, deployment and execution of services. This focus now has been changed into *adaptability*.

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

This thesis has focus on adaptability aspects related to capabilities. A capability is defined here as an inherent property of a node, which is used as a basis for implementing a service.

Goal adaptability properties of an adaptable service system can be classified as general and core properties. The general properties are requirement to the architectural framework while the core properties are requirement to the functionality. The core properties are classified as i) *re-arrangement flexibility*, ii) *failure robustness* and iii) *resource load awareness and control* properties.

The realization of the goal adaptability properties both needs an appropriate architectural framework as well as management functionality. This thesis presents a solution framework for reasoning-based capability configuration management for adaptable service systems. This framework is defined to consist of five contributions. Each contribution consists of sub-contributions; each of which represents contributed concept, model or mechanism. The contributions are:

- **C1:** Capability-based computing architecture
- **C2:** Policy-based reasoning
- **C3:** Capability configuration management

- **C4:** Concept model and data representation
- **C5:** Scenarios - experimentation and simulation

Capability-based computing architecture is a capability and QoS-based architectural framework intended to be used for the specification and execution of any service functionality. *Policy-based reasoning* is a support functionality that makes adaptable service systems being able to take decisions based on flexible and expressive behavioral specification. *Capability configuration management* is a functionality related to capability specifications, configuration, allocation, re-allocation and optimization. *Concept model and data representation* is the data model applied for the formalization and representation of the concepts applied for the capability configuration management based on policy-based reasoning. *Scenarios – experimentation and simulation* shows the experiments and simulations that have been conducted for validating the other contributions.

My PhD work and thesis is related to TAPAS (Telematics Architecture for Play-based Adaptable Service Systems). This thesis is structured into two main parts: Part I – Introduction and Part II – Selected publications. Part I is intended for the reader to get an overview of the publications included in Part II.

Preface

This thesis is submitted as partial fulfillment of the requirements for obtaining the Doctor of Philosophy degree (Ph.D.) at the Department of Telematics (ITEM), Norwegian University of Science and Technology (NTNU), Trondheim, Norway. Funded by NTNU, the work presented in this thesis has been worked out in the period of January 2003 – November 2007 under the supervision of Professor Finn Arve Aagesen. The work has been related to the Telematics Architecture for Play-based Adaptable Service System (TAPAS) project. TAPAS is a research project intended to study architecture concepts, which were initially related to dynamic Plug-and-Play (PaP) but now related to adaptable service systems. Presently the aim of the project is not only to specify, develop and perform experiment with parts of architecture concepts for adaptable service systems but also to find out how to increase the efficiency of, and to simplify installation, deployment, operation, management, maintenance and evolution of adaptable service systems. The project has been divided into several tasks. This thesis is associated with the concepts *capability*, *QoS*, *performance* and *reasoning* utilized in adaptable service systems.

The main part of this thesis consists of nine main publications, seven of which were submitted, published and presented in the proceeding of international conferences. One paper is being prepared for submission. One is published as a technical report. The papers have been written in collaboration with other researchers, mainly my supervisor. In the papers that I'm the first author, I have contributed to all parts of the paper, including the definition of research hypothesis, defining the architectural concepts, creating the formalisms and developing/performing the simulation- and/or analytical results, discussing the results, evaluating the research hypothesis, writing the papers, and presenting the paper. However, in Paper B and G, where I'm the second author, my contributions include: the creation of the formalisms, developing/performing the simulation- and/or analytical results, discussing the results, evaluating the research hypothesis and writing the paper.

Acknowledgements

Several persons have contributed to the success of my study and the smoothness of my stay at Norwegian University of Science and Technology (NTNU), Trondheim, Norway. First of all, I would like to gratefully acknowledge my supervisor Professor Finn Arve Aagesen at the Department of Telematics (ITEM). Finn Arve has always been a person with full understanding, care, patience and available to give me helpful supports whenever I needed. I'm convinced that his advises and comments have significantly enhanced the quality of my work. I would also love to thank Jittima Charoenpanich who made me want to pursue the Ph.D. for the first time.

As a perfect stranger to the kingdom of Norway in 2003, I would like to spread my sincere gratitude to the people who have always helped me neatly settle down during my stay. These people include, but not limited to, my supervisor himself, the Thai society in Trondheim, my Norwegian language teacher and classmates and people that I have been living and spending time with. Without them I wouldn't be able to adapt myself easily to the new place, new climate, new people, new food, new culture and new language.

My thankful gratitude goes to NTNU for funding my financial support, and to the people in TAPAS projects including Prof. Finn Arve Aagesen, Mazen Malek Shiaa, Shanshan Jiang, Cyril Carrez and Patcharee Thongtra as well as the other members of the *networked system working research group* at ITEM that have given me constructive comments and suggestions throughout my five-years research period. Special thanks to Mazen Malek Shiaa for being my supportive roommate and to Randi Schrøder Flønes and Mona Nordaune for helping me with all administrative tasks I'm not and will never be familiar with, Pål Sturla Sæther, Jarle Kotsbak and Asbjørn Karstensen for being the most talented and understanding technical team I have ever met.

Thanks to my family and my girl friend for being very supportive throughput the Ph.D. period. Last, but not least, I would love to express my deep gratitude to Venerable Dhammajayo Bhikku for always teaching and reminding me the important truth of life and that there are more important things to do in my life than writing the thesis.

Contents

List of figures

List of tables

List of publications

The publications constituting Part II of this thesis are listed in Table 1. Table 2 shows additional papers published as a part of my doctoral work, but that is not included in this thesis.

Table 1 – Overview of the papers included in Part II of this thesis

Paper A [Sup04]	An Approach to Capability and Status Modeling P. Supadulchai, F.A. Aagesen, in Proceedings of *Norwegian Informatics Conference (NIK 2003)*, Stavanger, Norway, 2004.
Paper B [Aag05]	Configuration Management for Adaptable Service Systems F.A. Aagesen, P. Supadulchai, C. Anutariya, M.M. Shiaa, in Proceedings of *IFIP International Conference on Metropolitan Area Network, Architecture, Protocols, Control and Management (MAN 2005)*, Ho Chi Minh City, Vietnam, 2005.
Paper C [Sup05a]	A Framework for Dynamic Service Composition P. Supadulchai, F.A. Aagesen, in Proceedings of *1st International IEEE Workshop on Autonomic Communication and Computing (ACC 2005)*, Taormina, Italy, 2005.
Paper D [Sup05b]	Autonomic Service Configuration by a Combined State Machine and Reasoning Engine-based Actor P. Supadulchai, F.A. Aagesen, in Proceedings of *the 2005 IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM 2005)*, Delta Centre-Ville Hotel, Montréal, Canada, 2005.
Paper E [Sup07a]	Policy-based Adaptable Service Systems Architecture P. Supadulchai, F.A. Aagesen, in Proceedings of *the IEEE 21st International Conference on Advanced Information Networking and Applications (AINA-07)*, Niagara Falls, Canada, 2007.
Paper F [Sup07b]	Towards Policy-Supported Adaptable Service Systems P. Supadulchai, F.A. Aagesen, in Proceedings of *the 13th Eunice Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services*, University of Twente, the Netherlands, 2007.

Paper G [Aag07]	A Capability-based Service Framework for Adaptable Service Systems F.A. Aagesen, P. Supadulchai, in Proceedings of *The 2nd International Conference on Advances in Information Technology (IAIT2007)*, Bangkok, Thailand, 2007.
Paper H [Sup07c]	Autonomous Production of Parameters of an Autonomous Capability Allocation Adaptation Model P. Supadulchai, F.A. Aagesen, prepared for a submission.
Report I [Sup07d]	NxET Reasoning Engine P. Supadulchai, *Plug-and-play Technical Report*, Department of Telematics, NTNU, ISSN 1500-3868.

Table 2 – Overview of the papers published but not included in this thesis.

Paper X	A Dynamic Configuration Architecture F. A. Aagesen, C. Anutariya, M. M. Shiaa, B. E. Helvik and P. Supadulchai, in Proceedings of *IEEE/IFIP Network Operations and Management Symposium (NOMS'2004)*, Seoul, South Korea, 2004.
Paper Y	An XML based Framework for Dynamic Service Management M.M. Shiaa, S. Jiang, P. Supadulchai and J. J. Vila-Armenegol, in Proceedings of *IFIP International Conference, INTELLCOMM 2004*, Bangkok, Thailand, 2004.

Part I – Introduction

1. Overview

1.1 Adaptable Service Systems – Definitions and Goal Properties

N*etworked services* are considered. *Networked services are offered by service systems* consisting of service components, which by their inter-working, provide a service in the role of a service provider to a service user. Service users can be human users as well as software and/or hardware components. Service components are executed as software components in nodes, which are physical processing units such as servers, routers, switches, PCs and mobile phones. A *service framework* is defined here as the overall structural and behavior framework for service specification and service execution. *Service specification* concerns how to formally define the structure and the behavior of a service system, which constitutes the service. *Service execution* is the execution of the implementation of the defined service specifications.

During more than two decades, *networked service* systems have been an important research topic. Example topics include Intelligent Networks [Ino99], TINA (Tele-communication Information Networking Architecture) [ITU92], Mobile Agents [Woo02], and Active and Programmable Networks [Ten97] in the 80-ies and 90-ies. Topics in the late 90-ies include Pervasive and Ubiquitous Computing [Lew04]. Today's topics include Web Services [Alo03] and Autonomic Systems and Communication [Dob06]. Focus was on efficiency in the definition, deployment and execution of services. This focus now has been shifted into *adaptability* – how networked service systems can adapt to changes in their environments. We have entered an era with a higher degree of flexibility. To utilize the potential of these features, the properties of services, nodes and node capabilities must be formalized, stored and made available. Also, the policies and mechanisms that manipulate and exploit these properties must be worked out.

Adaptability is a concept widely used in Biology to describe the agility of biological entities that are able to adapt to changes in the environment [Sus05].

Adaptability in service systems is defined in a similar way, however, with a different perspective.

Adaptable service systems are in Paper H defined as service systems that are able to adapt dynamically to changes in time and position related to users, nodes, capabilities, system performance, changed service requirements and policies.

A capability is defined here as an inherent property of a node, which is used as a basis for implementing a service. Capabilities can be classified as *resources*, *functions* and *data*. Examples of general capabilities are CPU, memory, transmission link, available programs and user profiles. Examples of specific capabilities are medical equipment related to networked medical services.

Capability performance measures are in Paper H classified as i) Capacity measures, ii) State measures and iii) QoS measures. *Capacity measure* examples are transmission channel capacity, CPU processing speed and disk size. *State measure* examples are the number of available streaming connections and the number of packets discarded in a specific buffer. *Quality of Service (QoS)* is the degree of satisfaction of the users of a service. *QoS measures* can be traffic measures and dependability measures. Important *traffic measure* examples are capability throughput and capability utilization. Important *dependability measure* examples are capability availability and capability recovery time.

Service performance measures are measures related to the service. These can comprise i) state measures and ii) QoS measures. *State measure* examples are the number of users waiting to use the service, the number of users that are using the service and the status of the service; i.e. normal state, performance-degraded state. Concerning *QoS measures*, *traffic measure* examples are service connection time, service transfer time, service waiting time, service throughput and service utilization. Important *dependability measure* examples are service availability and service recovery time.

The concept *system performance* is used as a common concept for capability and service performance. As defined in Paper G, The service provider and service users can have common agreements related to the provided functionality, capabilities, system performance as well as payment. Such an agreement can be expressed in a Service Level Agreement (SLA). *SLA is a formal negotiated agreement between a service provider and service users* [Try05].

To precisely describe the properties of adaptable service systems, *goal adaptability properties* consisting of general and core properties are defined. The general properties are properties of the *architectural framework*, while the core properties are properties of the *functionalities*.

General properties are in Paper G defined as follows: 1) There must be a flexible and common way of modeling 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 defined in Paper B and later in Paper G are illustrated in Figure 1.The core properties are grouped into three classes:

A1:	Rearrangement flexibility
A2:	Failure robustness
A3:	Resource load awareness and control

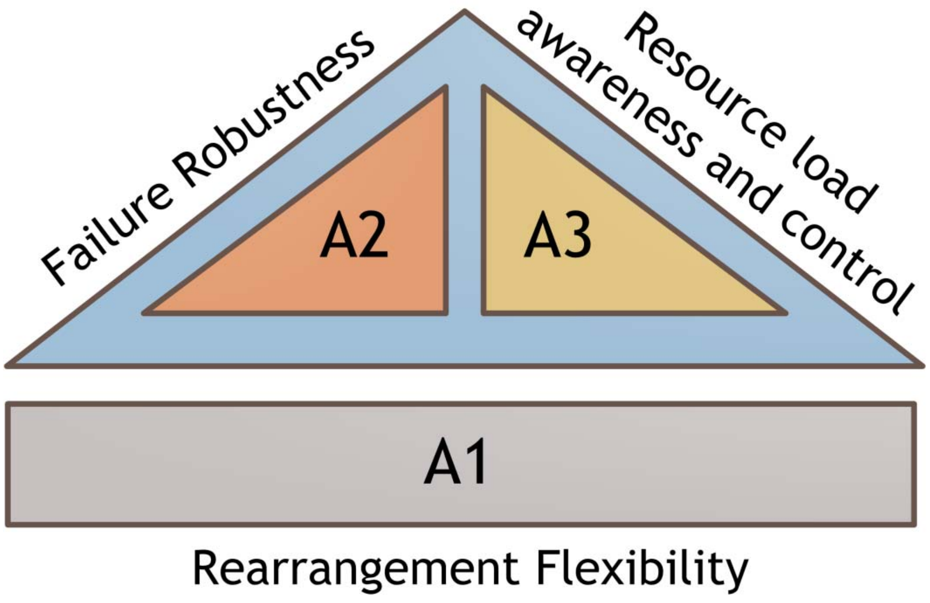

Failure Robustness
Resource load awareness and control
A2
A3
A1
Rearrangement Flexibility

Figure 1 – The core properties **A1**, **A2** and **A3**

Rearrangement flexibility means that the system structure and the functionality are not fixed. Nodes, users, services, service components, capabilities, can be added, moved, removed according to needs. Mobility of persons, sessions, nodes, terminals is further seamlessly handled. New nodes and capabilities are found automatically when introduced, and needed information about changes is propagated. There is a continuous adaptation to changed environments and operation strategies/policies.

Failure robustness means that the architecture is dependable and distributed, and that the system can reconfigure itself in the presence of failures. Resources and functionality are duplicated, hardware and software component failures must be detected and reconfiguration and re-initialization must take place during system operation.

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

The general properties as well as property **A1** define the basic flexibility features of the architecture. Property **A2** and property **A3** set requirements to concepts and features that are needed as a foundation for the specification and implementation of the functionalities defined by these properties. The core properties will, from now on, be referred as **A1**, **A2** and **A3** throughout this thesis.

1.2 Outline of the thesis

The thesis is structured into two main parts: **Part I** – Introduction and **Part II** – Selected Publications. **Part I** is furthered structured as follows:

Section 1 presents definitions and goal properties of adaptable systems, an outline of the thesis as well as a guideline for reading. Section 2 presents research method and framework. This includes objectives, scope, problem statements, a presentation of TAPAS, and the research cycle of the work presented in this thesis. The research cycle visualizes the research methodology, the evolution of this research and the relations with the TAPAS architecture. Section 3 presents the solution framework, the contributions and finally the realization of the problem statements defined in Section 2. Section 4 gives a summary of the papers of **Part II** as well as the relationship between them. Section 5 presents the related work. Section 6 presents summary, conclusions and further work.

The publications included in **Part II** are the followings:

Paper A An Approach to Capability and Status Modeling

P. Supadulchai, F.A. Aagesen, in Proceedings of *Norwegian Informatics Conference (NIK 2003)*, Stavanger, Norway, 2004.

Paper B Configuration Management for Adaptable Service Systems

F.A. Aagesen, P. Supadulchai, C. Anutariya, M.M. Shiaa, in Proceedings of *IFIP International Conference on Metropolitan Area Network, Architecture, Protocols, Control and Management (MAN 2005)*, Ho Chi Minh City, Vietnam, 2005.

Paper C A Framework for Dynamic Service Composition

P. Supadulchai, F.A. Aagesen, in Proceedings of *1st International IEEE Workshop on Autonomic Communication and Computing (ACC 2005)*, Taormina, Italy, 2005.

Paper D Autonomic Service Configuration by a Combined State Machine and Reasoning Engine-based Actor

P. Supadulchai, F.A. Aagesen, in Proceedings of *the 2005 IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM 2005)*, Delta Centre-Ville Hotel, Montréal, Canada, 2005.

Paper E Policy-based Adaptable Service Systems Architecture

P. Supadulchai, F.A. Aagesen, in Proceedings of *the IEEE 21st International Conference on Advanced Information Networking and Applications (AINA-07)*, Niagara Falls, Canada, 2007.

Paper F Towards Policy-Supported Adaptable Service Systems

P. Supadulchai, F.A. Aagesen, in Proceedings of *the 13th Eunice Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services*, University of Twente, the Netherlands, 2007.

Paper G A Capability-based Service Framework for Adaptable Service Systems

F.A. Aagesen, P. Supadulchai, in Proceedings of *The 2nd International Conference on Advances in Information Technology (IAIT2007)*, Bangkok, Thailand, 2007.

Paper H Autonomous Production of Parameters of an Autonomous Capability Allocation Adaptation Model

P. Supadulchai, F.A. Aagesen, prepared for a submission.

Report I NxET Reasoning Engine

P. Supadulchai, *Plug-and-play Technical Report*, Department of Telematics, NTNU, ISSN 1500-3868.

1.3 Guideline for reading

The publications presented in **Part II** are self-contained. They are reformatted to improve readability when published in a smaller-size book. Since the papers have some overlapped contents and can also be linked, the reader is suggested to follow the depicted sequence illustrated in Figure 2.

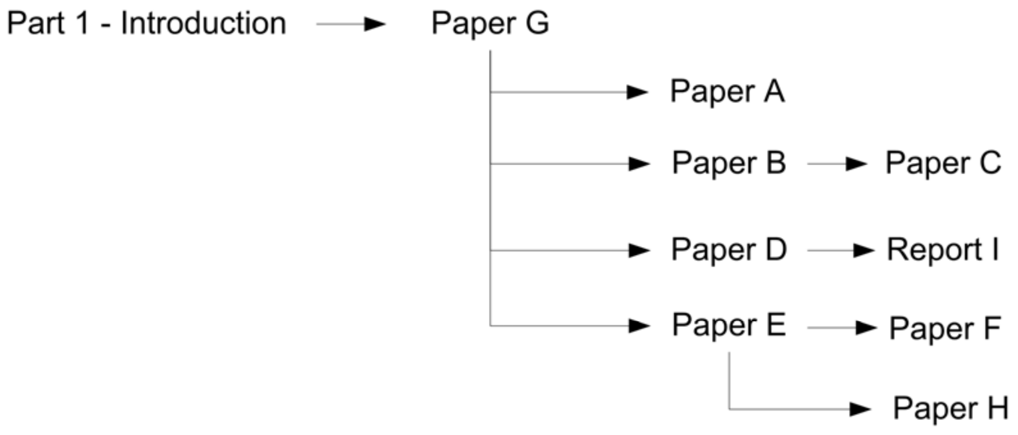


Figure 2 – Guidelines for reading

2. Research method and framework

This section describes the research objectives, problem statements, TAPAS as well as the research cycle of the work presented in this thesis. Section 2.1 states the overall research objectives. Section 2.2 presents the problem statements. Section 2.3 presents TAPAS, while the research cycle is presented in Section 2.4.

2.1 Research objectives

The context of this thesis is capabilities in adaptable service systems. The main research objectives are *"to study and analyze concepts, models and mechanisms for the realization of the general and the core properties related to capabilities, and further to specify, construct, evaluate and validate a framework for capability configuration management."*

2.2 Scope

Concerning the design of the capability configuration management framework, the emphasis is not on distribution and dependability aspects. The capability configuration management functionality is not itself designed to meet the core property A2 or to support the design of the core property A2 besides needed concepts. The capability configuration management functionality is defined to support the core properties A1, A2 and A3, but its design is based on a centralized and non-dependable architecture.

2.3 Problem statements

The following *initial problem statements* **P1, P2, P3, P4, P5** and **P6** are defined.

- **P1:** How to include capabilities as well as additional needed concepts in the architectural framework?
- **P2:** How to include capability management in an architectural framework?

- **P3:** Which concepts, features and mechanism are appropriate for autonomic capability management?
- **P4:** What is the ontology to be used? How to model and represent the concepts of the ontology?
- **P5:** How to model capability management functionality when considering modularity, reusability, expressive power and flexibility?
- **P6:** How to evaluate and validate the proposed framework?

2.4 TAPAS

The work in this thesis is related to TAPAS architecture (TAPAS = Telematics Architecture for Play-based Adaptable Service Systems). The original architecture has focus on the adaptability properties of plug-and-play systems [Aag99a]. Since then several participants have contributed to the concepts, designs, implementations and demonstrations [Shi05]. The PhD work presented in this thesis has also contributed with new TAPAS concepts, models and mechanisms. TAPAS as it was by the time this PhD study commenced has been improved and extended. This section presents the scope of the thesis considered from the TAPAS perspective. Necessary features and concepts of the TAPAS architecture will be presented. Contributions from the work reported in this thesis will also be given.

2.4.1.1 TAPAS computing and service architectures

To meet the general properties as defined in Section 1, the TINA architecture principle [ITU92] was followed. TAPAS has a computing architecture and a service architecture dimensions.

The service architecture has focus on the service functionality independent of implementation. The computing architecture has focus on the modeling of functionality with respect to implementation, but independent of the nature of the service functionality. The properties of the computing architecture, however, are the fundament for the creation of services with needed goal adaptability properties.

TAPAS computing architecture

The computing architecture is founded on a *theatre metaphor* as illustrated in Figure 3. *Actors* perform *roles* according to *manuscripts*. Actors are software components in the nodes that can download manuscripts defining the roles to be played. An actor will constitute a *role figure* by behaving according to a manuscript. A *play* consists of several *actors* playing different *roles*, each possibly having different requirements on *capabilities*.

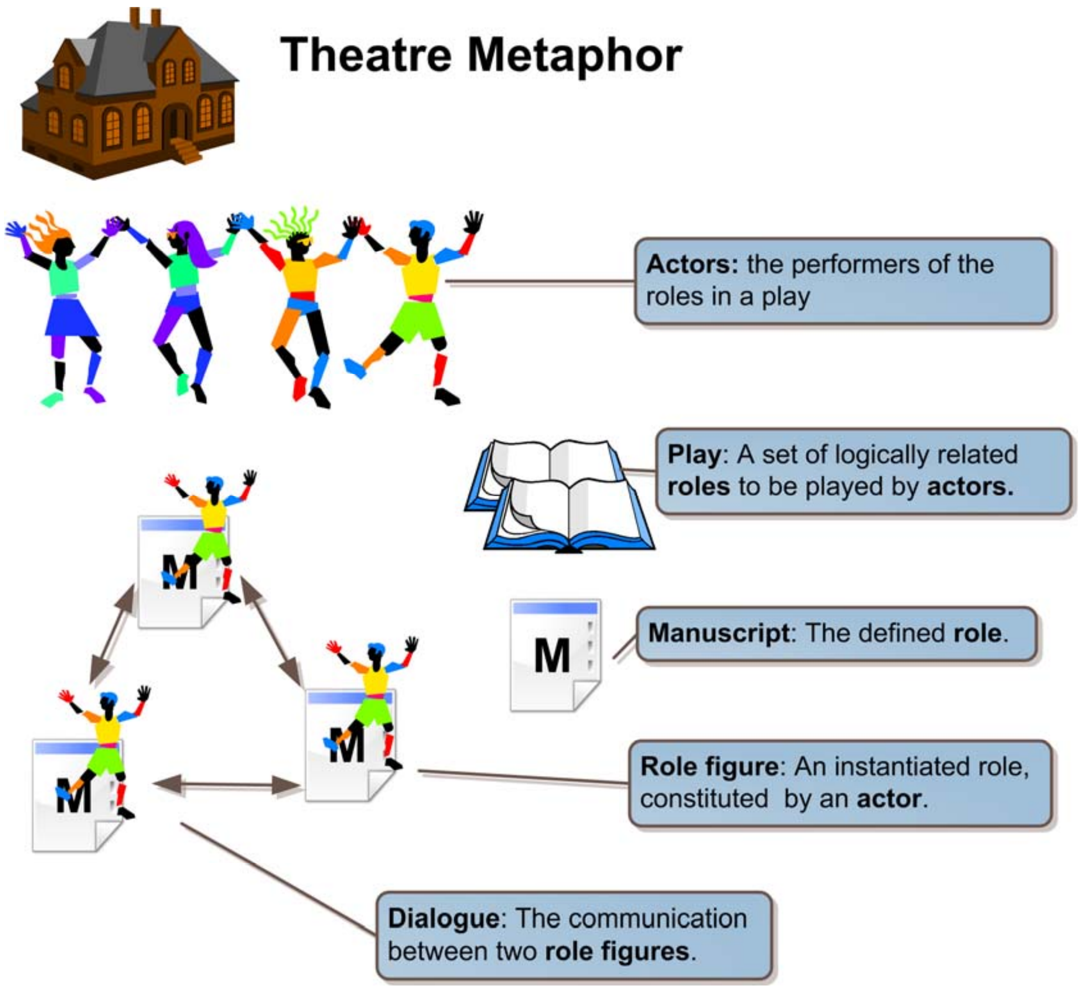

Theatre Metaphor
Actors: the performers of the roles in a play
Play: A set of logically related **roles** to be played by **actors.**
M
Manuscript: The defined **role**.
Role figure: An instantiated role, constituted by an **actor**.
Dialogue: The communication between two **role figures**.

Figure 3 – The theatre metaphor

Figure 9 in Section 3.1 illustrates the present version of the TAPAS computing architecture. The computing architecture has three views: the *service view*, *play view* and *physical view*. In the service view, a service system providing a service is constituted by conceptual *service components*, each of which can also be considered as a service system and can be further constituted by service components. A service component that cannot be further de-composed is denoted as a leaf service component.

Leaf service components in the service view are constituted by role figures. A role figure can have dialogues with other role figures. A play consisting of role figures and dialogues is a service system instance as seen from the play view.

Considering the physical view, actors are executed as operating system software components in nodes, which are physical processing units such as servers, routers, switches and user terminals. The *core platform* is a platform supporting the execution of service functionality based on the computing architecture functionality. The core platform has a manuscript execution functionality and a basic communication functionality between nodes. In the original version denoted as the classical actor model, a manuscript is based on Extended Finite State Machine (EFSM). The communication functionality can be based on Java RMI [Aag01a], socket-based [Lüh04] and XML web services [Vil04].

The three view model is not a layered service model. A service system and its service components have models and model parameters related to the service view, play view and physical view. The visibility of physical view parameters in the service view is a design decision rather than a model restriction.

TAPAS service architecture

The service architecture is illustrated in Figure 4. In addition to the *primary service provisioning functionality* for application service systems, there must be functionality for management of the networked service systems. This *management functionality* comprises:

- *Service management:* Service specification, implementation, deployment, invocation, exhibition, discovery and access.
- *Mobility management:* The handling of mobility and mobility components. Mobility components can be persons, services, dialogues and sessions, terminals, nodes, capabilities, data and programs.
- *Capability and service performance administration:* Monitoring and registration of available and allocated capabilities, monitoring and registration of system performance, and the provisioning of a repository of available capabilities, capability performance and service performance.
- *Capability configuration management:* The allocation, re-allocation, de-allocation and optimization of the use of capabilities.

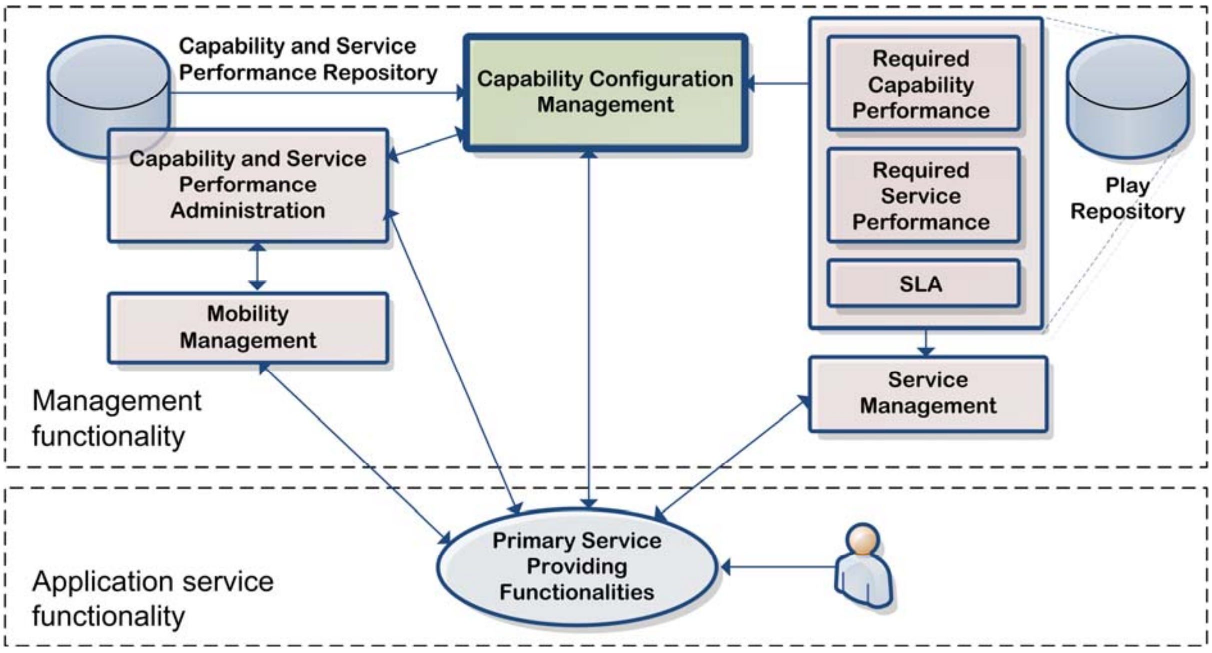

Capability and Service Performance Repository
Capability Configuration Management
Required Capability Performance
Required Service Performance
SLA
Play Repository
Capability and Service Performance Administration
Mobility Management
Service Management
Management functionality
Primary Service Providing Functionalities
Application service functionality

Figure 4 – TAPAS Service architecture

Regarding the service architecture, *this thesis focuses on and has contribution mainly to capability configuration management.* The studies of the other functionality parts of the adaptable service system can be found in the related work by the other participants in the TAPAS project. A comprehensive study on the mobility management aspects can be found in Mazen Malek Shiaa's thesis [Shi05]. A study on the service management aspects can be found in Shanshan Jiang's work [Jia03, Jia06, Jia07], which is a part of an on-going PhD study.

2.4.1.2 This thesis's contributions to TAPAS

At the starting point of this PhD study, the architecture had a core platform, which is a generic software component for executing the so-called Extended Finite State Machine (EFSM)-based specifications. In addition to the core platform, the major service functionalities that had been contributed to the architecture were related to the mobility management functionality [Shi05]. The architecture itself had limited features related to the capability concepts.

The capability concept was defined in [Aag99b] and a configuration management framework for capability configurations was introduced in [Aag02a] and [Aag02b]. However, the configuration management functionality was limited.

The work in this thesis has contributed concepts, models and mechanisms to the TAPAS architecture. The computing as well as the service architecture has been extended and revised. In addition to the revisions and improvement of the concept model constituting the computing architecture, new concepts such as capability performance, service performance, SLA, Reasoning Machine (RM) and RM manuscript have been added. The core platform has also been improved with the ability to execute policy-based specifications, which is more flexible and powerful than EFSM-based specifications. The RM applied in the core platform, which is more thoroughly described in Report I, is one of my contributions to the core platform. The other main contribution to the core platform is the concept model and data representation as is described in Section 3.

Concerning the service architecture, the capability configuration management is extended from the configuration management framework in [Aag02b] and [Aag03] with respect to capability performance, service performance and SLA. In addition to extending the functionality area of capability configuration management, policy-based adaptability models based on feedback loops are introduced. The RMs handling the adaptation is controlled by policies and parameters. Policies are rules and actions, while the parameters are constraints as well as reasoning conditions (See the RM-related data entities in Section 3). Dynamic policies allows policies used by the capability allocation adaptation to be changed by another policy-based system based on some goodness criteria. Parameter production allows parameters of the capability allocation adaptation to be re-produced based on some goodness criteria.

The complete solution framework and its contributions will be given in Section 3. However, a brief comparison between TAPAS at the starting point of this thesis (TAPAS 2003) and the present TAPAS architecture (TAPAS 2007) is illustrated in Figure 5 and Figure 6. Figure 5 compares the TAPAS computing architecture in 2003 and 2007 with respects to the capability, capability performance, service performance, SLA concepts and the core platform. Figure 6 compares the TAPAS service architecture in 2003 and 2007 with respects to the capability configuration management and adaptation models.

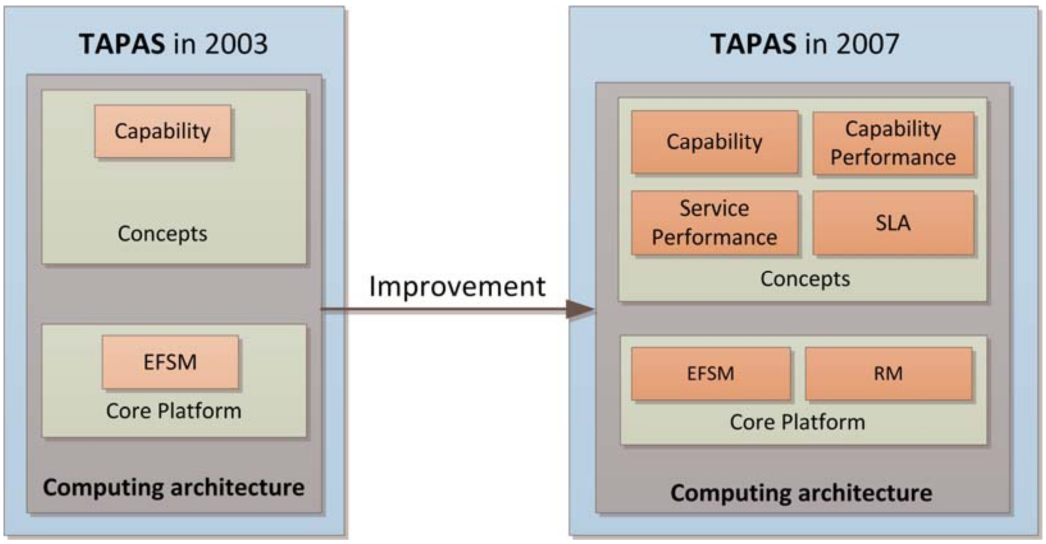


Figure 5 - TAPAS computing archtecture in 2003 and 2007

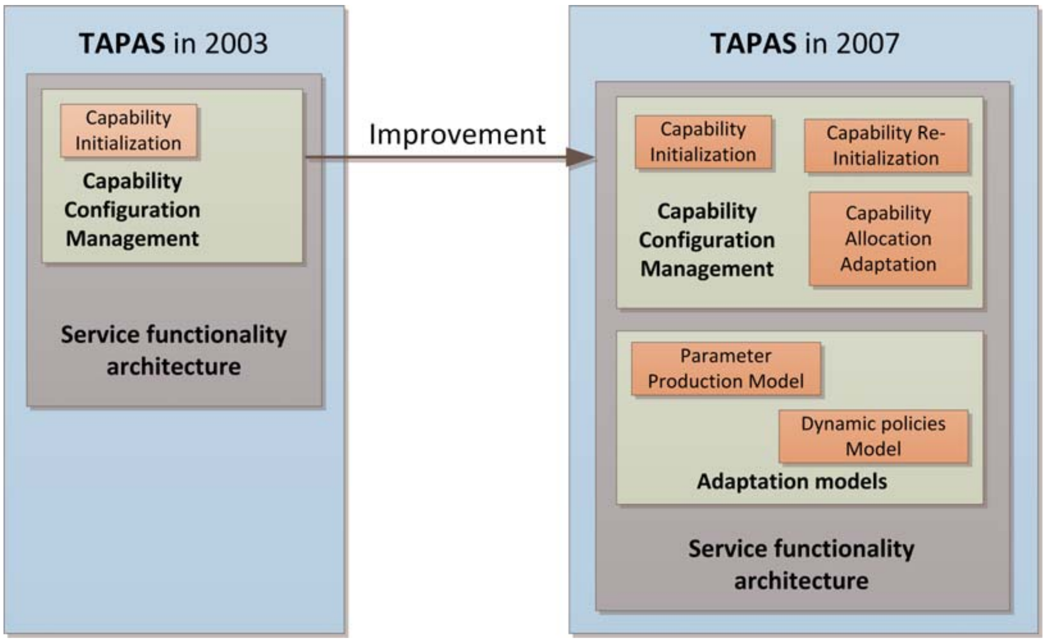

TAPAS in 2003
Capability Initialization
Capability Configuration Management
Service functionality architecture
Improvement
TAPAS in 2007
Capability Initialization
Capability Re-Initialization
Capability Configuration Management
Capability Allocation Adaptation
Parameter Production Model
Dynamic policies Model
Adaptation models
Service functionality architecture

Figure 6 - TAPAS service architecture in 2003 and 2007

2.5 Research cycle

The research cycle used for the work in this thesis is based on common scientific iterative research cycle method [Bri91] and [Aag01b] as follows:

Solution proposition

Concepts, *models* and *mechanisms* are proposed for the problem statements as defined in 2.3.

Modeling and formalization

A solution framework consisting of the proposed concepts, models and mechanisms is formulated. The concepts, models, mechanisms are further formalized to ensure integrity, soundness, consistency and completeness. Using formalized concepts, models and mechanisms, it is easier to improve and add new functionality than using the non-formalized ones.

Evaluation and validation

Demonstration and *simulation* have been applied for validating and improving the proposed solution framework. *Demonstration* is a method for illustrating how the proposed models and mechanisms perform in a real environment. *Simulation* is a method for testing the proposed models and mechanisms. The demonstration method has been used when the system performance evaluation is not the main focus. Simulations have been used when real-environment demonstrations have not been satisfactory and when the applications of the system models were needed to give more detailed results with respect to system performance. Simulations can also be easily reproduced as opposite to real-scale demonstrations.

Concerning the demonstration method, Paper A, Paper B, Paper C and Paper D present several scenarios for adaptable service systems. The formalized and executable specifications in these scenarios are demonstrated by the solution framework. Demonstrations illustrate the execution mechanism and validate the obtained results.

Concerning the simulation method, Paper E, Paper F and Paper H presents the simulation results based on the proposed models and mechanisms. The simulation method is preferred because QoS and performance of the service systems in the scenarios are the main focus. Model simulation is more feasible and efficient than large scale real-environment demonstrations.

There is no formal proof of the models and mechanisms. The validation is done by the demonstration and simulation methods.

Result discussion

Results, which are gathered from the evaluation and validation, are used for further discussing the models and mechanisms. If the results are not as expected, the models and mechanisms will be improved or some assumptions will be changed. This may subsequently lead to a new problem statement or a new solution and begin a new iterative research sequence.

Figure 7 illustrates an overview of the research cycle used in this thesis, which consists of solution proposition, modeling and formalization, evaluation and validation and result discussion.

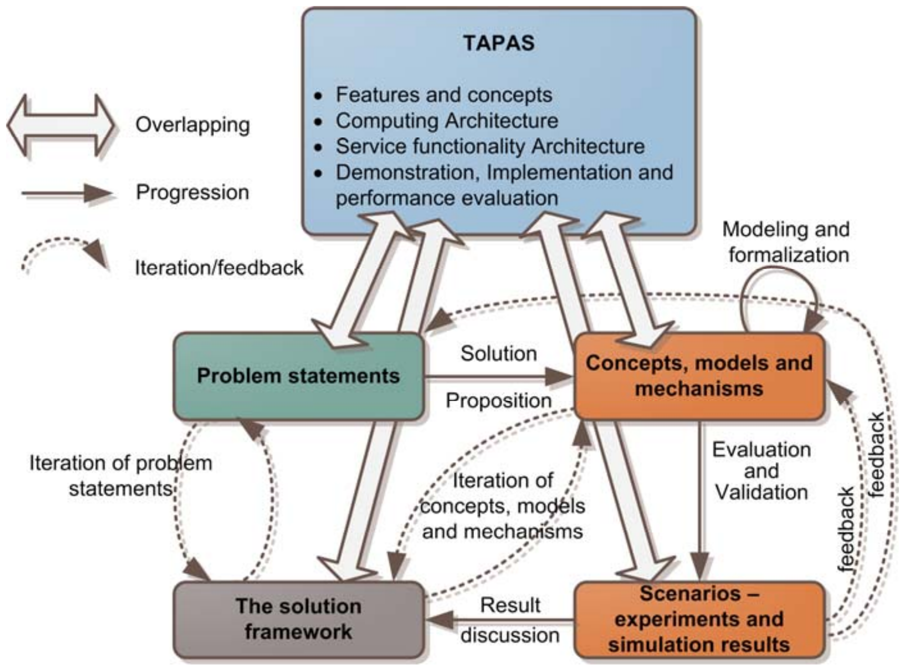

Overlapping
Progression
Iteration/feedback
TAPAS
Features and concepts
Computing Architecture
Service functionality Architecture
Demonstration, Implementation and performance evaluation
Modeling and formalization
Problem statements
Solution
Proposition
Concepts, models and mechanisms
Iteration of problem statements
Iteration of concepts, models and mechanisms
Evaluation and Validation
feedback
feedback
The solution framework
Result
discussion
Scenarios – experiments and simulation results

Figure 7 – Research cycle overview

3. The solution framework

The solution framework is defined to consist of four contributions. Each contribution consists of sub-contributions; each of which represents contributed concept, model or mechanism. The contributions are:

C1: Capability-based computing architecture

C2: Policy-based reasoning

C3: Capability configuration management

C4: Concept model and data representation

C5: Scenarios - experimentation and simulation

Section 3.1 presents the contributions. The relationship between contributions and the problem statements defined in Section 2.3 is discussed in Section 3.2.This section also presents the relationship between the contributions and the publications in **Part II**.

3.1 *The solution framework contributions*

Figure 8 illustrates the sub-contributions of the four contributions **C1-C4**. These contributions are not independent. The ontology-based capability selection mechanism, dynamic policies and parameter production, for examples, are related to both policy-based reasoning and capability configuration management. The content of the contributions defined by their sub-contributions are presented in more details below. The contribution **C5** contains the applications of the solution framework. Thus C5 is not included in Figure 8.

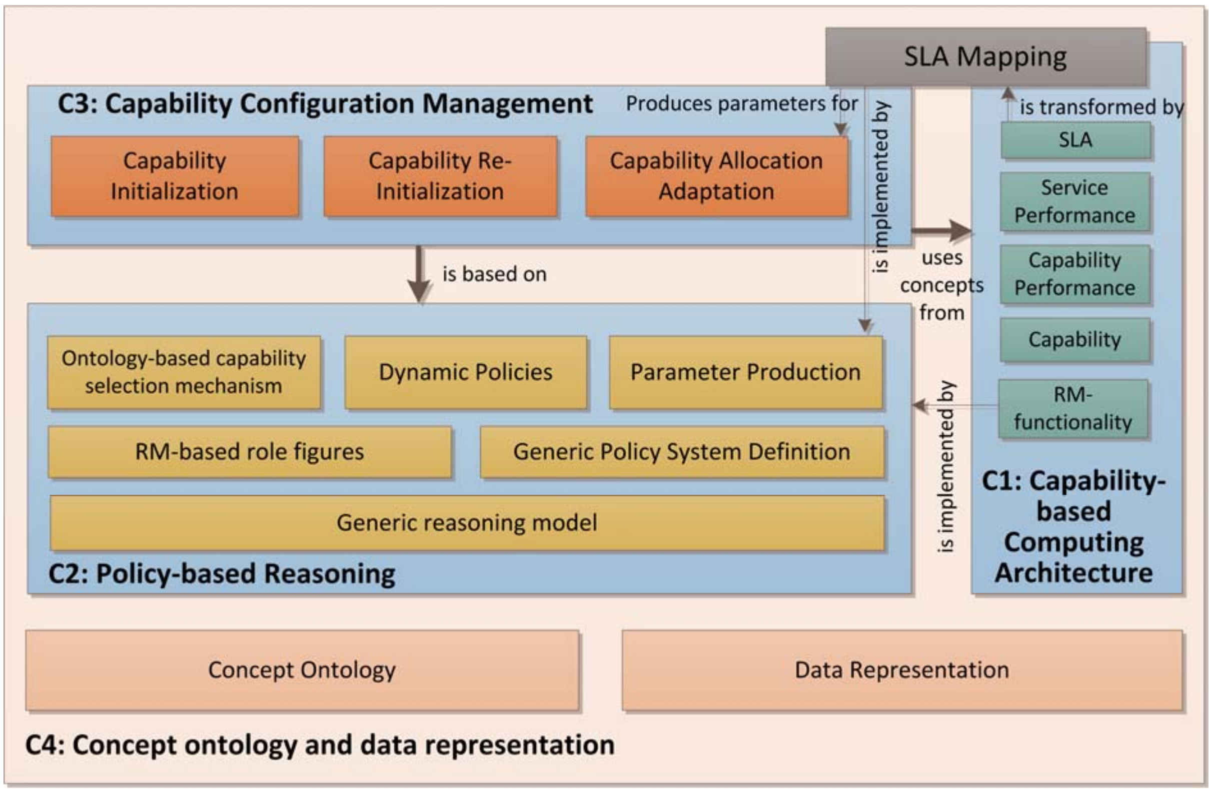

SLA Mapping
C3: Capability Configuration Management
Produces parameters for
is implemented by
is transformed by
Capability Initialization
Capability Re-Initialization
Capability Allocation Adaptation
SLA
Service Performance
uses concepts from
Capability Performance
is based on
Capability
Ontology-based capability selection mechanism
Dynamic Policies
Parameter Production
RM-functionality
RM-based role figures
Generic Policy System Definition
is implemented by
Generic reasoning model
C1: Capability-based Computing Architecture
C2: Policy-based Reasoning
Concept Ontology
Data Representation
C4: Concept ontology and data representation

Figure 8 - The reasoning-based capability configuration management framework

C1: Capability-based computing architecture

Capability, capability performance, service performance, SLA and RM-functionality

The capability-based computing architecture, which is illustrated in Figure 9, extends the previous versions of the TAPAS computing architecture. The important roles of capabilities are made clearly visible and QoS-related concepts such as: capability performance, service performance and SLA are also included. The core platform has also been improved with the ability to execute policy-based RM specifications. These concepts are formalized and made visible in the computing architecture as illustrated Figure 9.

The orange boxes in Figure 9 depict the contributed concepts. Capability and capability performance measures defined in the physical view are also visible in the play view as well as the service view. The service performance measures and SLA defined in the service view are also visible in the play view. This is not explicitly illustrated in Figure 9.

SLA classifies service users into different classes, each of which represents the agreed priority, functionality, performance, payment and penalties functions for a group of users with the same degree of satisfaction and costs. The capability allocation adaptation is the mechanism used to allocate and re-allocate capabilities with respect to the SLA defined.

SLA Mapping

SLA mapping is a transformation of SLA into the parameters of the capability allocation adaptation model. In addition to **C1**, SLA mapping is also related to the parameter production in **C2** and the capability allocation adaptation in **C3**. The parameters to be produced include the system constraints as well as the reasoning conditions of the RM-based role figure handling the capability allocation adaptation. The mapping is based on the parameter production that is handled by the parameter producer as illustrated in Figure 10. An example of the SLA mapping has been dealt in Paper H.

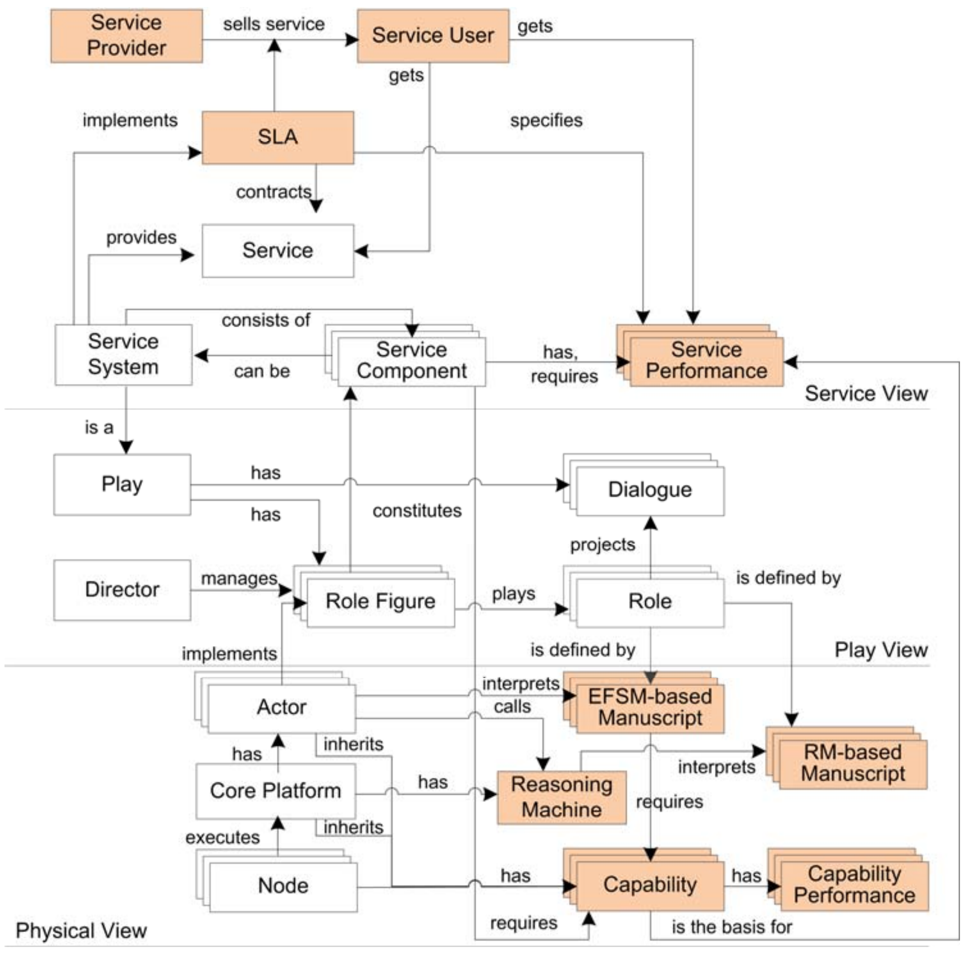


Figure 9 – The capability-based computing architecture

To further create satisfaction for the service provider, income and cost functions are considered. The income function determines instantaneous profits of the service provider while the cost function determines instantaneous cost of the service provider. The income and cost functions are also considered parameters of a capability allocation adaptation model. Examples can be found in Paper E, Paper F, Paper G and Paper H.

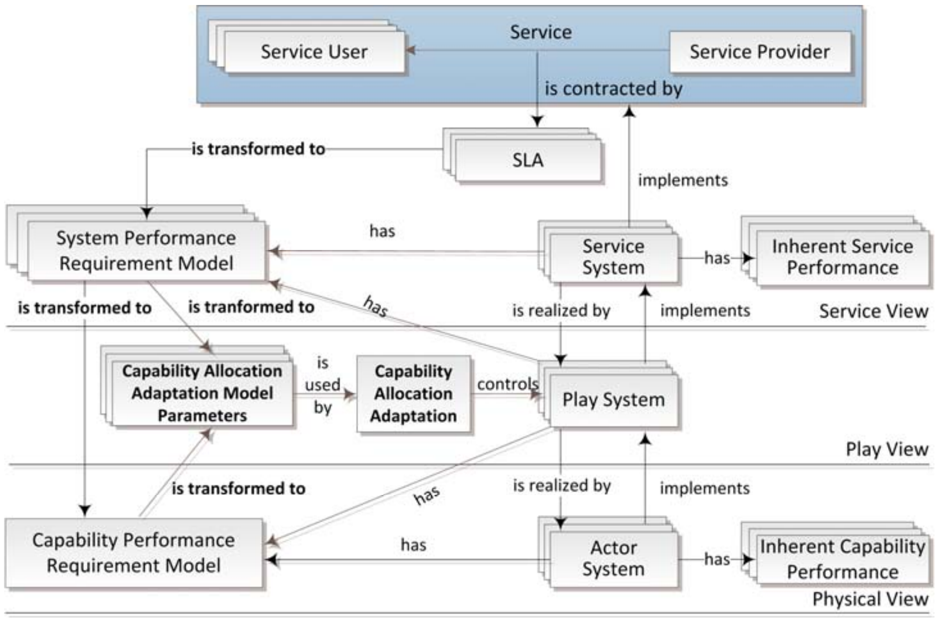


Figure 10 – SLA Mapping

C2: Policy-based reasoning

Policy-based reasoning is a support functionality related to the ability of adaptable service systems to take decision based on flexible and expressive behavioral specification. The contribution consists of generic reasoning model, RM-based role figures, generic policy system definition, ontology-based capability selection mechanism, dynamic policies and parameter production. Policy-based reasoning is a contribution to be found in all Papers. The formalization of the policy-based reasoning including the generic reasoning model and RM-based role figures can be found in Paper E, Paper F, Paper G and Paper H.

Generic reasoning model

The generic reasoning model has a *reasoning procedure* for transforming an initial transformation clause to final transformation clause(s), which are mediums for the transformation. The reasoning procedure uses policies, facts and system constraints. Policies are used to manage the behavior of service systems. A policy system consists of rules and actions. The rules are used to transform an initial transformation clause to final transformation clause(s), which can contain suggested actions. These actions will be applied for managing the adaptable service system behavior. The facts are inherent data gathered from the system. Examples are inherent capability performance, inherent service performance and SLA. The system constraints are used to model application-specific service system requirements with respect to capability, capability performance and service performance.

The generic reasoning model can be used to form a feedback loop. An example feedback loop is the capability allocation adaptation model, which will be described later in this section. Reasoning conditions are used by an EFSM-based role figure to activate and de-activate the feedback loop for the capability allocation adaptation. The feedback loop parameters are the system constraints and the reasoning conditions. These parameters can be produced.

Figure 11 illustrates the generic reasoning model. The system constraints, rules and actions can have *variables*. The result of the reasoning can, in addition to actions, give instantiated variables. The facts contain no variable.

The generic reasoning model has been used in all papers. The implementation of the reasoning procedure is discussed in Report I.

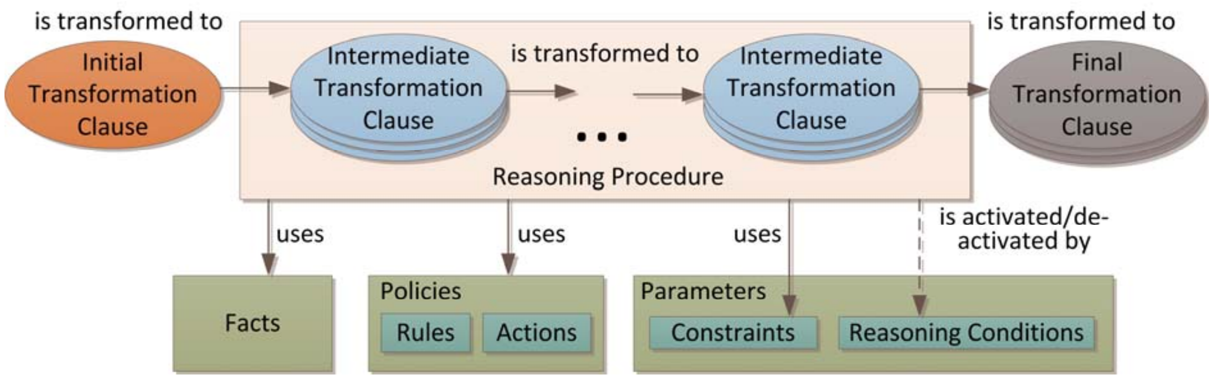

is transformed to
Initial Transformation Clause
Intermediate Transformation Clause
is transformed to
Intermediate Transformation Clause
is transformed to
Final Transformation Clause
Reasoning Procedure
uses
uses
uses
is activated/de-activated by
Facts
Policies
Rules
Actions
Parameters
Constraints
Reasoning Conditions

Figure 11 - The generic reasoning model

RM-based role figures

As introduced in Section 2, manuscripts are the specification of roles to be played by actors. An actor can have two types of manuscripts: the EFSM-based specification or the combination of the EFSM-based and RM-based specification. Concerning the role figures, there are also two role figure types: i) EFSM-based role figures and ii) RM-based role figures. The EFSM-based role figure has its role defined by an EFSM specification. The RM-based role figure has its role defined by the combination of an EFSM-based and the RM-based specification.

The functionality domain of RM-based role figures can be separated into two cases. In the first case, the RMs of a service system have no access to capability and service performance measures of the service system. In this case, the RM provides an ordinary procedural reasoning service only; i.e. the reasoning service does not include the capability configuration management. In the second case, RMs of the service system have access to capability and service performance data of the service system. These RMs take part in the capability configuration management for managing other–EFSM-based role figures. For the second case, RMs can be used as:

A. a procedural reasoning service for capability initialization and re-initialization based on capability performance measures

B. a feedback loop service for capability allocation adaptation based on capability and system performance measures

C. a feedback loop service constituting *dynamic policies* and *parameter production* for the capability allocation adaptation model in B

A reasoning cluster is an independent unit with respect to reasoning. It is a collection of EFSM-based service components with an associated *reasoning system* constituted by one or more *RM-based service components*. A reasoning cluster has associated RM-related data entities and considers them as common data shared among EFSM-based and RM-based role figures. In addition to the EFSM-based service components in the cluster, a reasoning machine has an associated cooperating EFSM, which will invoke the reasoning machine. Depending of the nature of the cluster and the nature of the reasoning, a dedicated EFSM denoted as E_{Σ} can be needed to inspect the reasoning conditions and to activate the RMs associated EFSM. Three reasoning cluster types (A, B and C) corresponding to the functionality domain cases A-C defined above are illustrated in Figure 12.

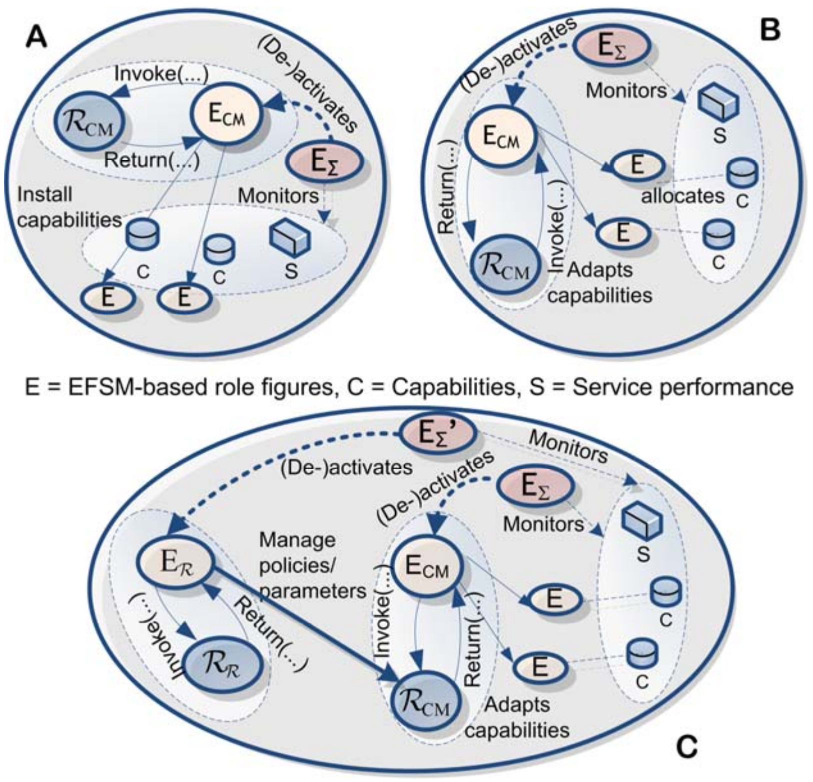


Figure 12 - Reasoning cluster types

In all cluster types, an RM-based *capability manager* $\mathcal{R}_{\mathrm{CM}}$, gives suggested actions to an EFSM-based *capability manager* E_{CM} for managing capabilities. In the cluster type A, $\mathcal{R}_{\mathrm{CM}}$ is a procedural service for capability initialization and re-initialization. Re-initialization is initiated by E_{Σ}.

Generic policy system definition

Generic policy system definition defines policies that are applicable for any adaptable service systems. Constraints specific to an individual application service system will be separated from policies. The separation makes the policies generic. However, a condition is required. A policy system to be made generic must represent behaviors that are independent of the nature of any service system. These generic behavior examples can be self-x autonomic functionality such as self-healing, self-protection and self-optimization.

Example of policy systems that have been made generic is the policy system for the capability initialization and the policy system for the parameter production. The generic policies system for capability initialization can be found in Paper C while the generic policy system for parameter production can be found in Paper H.

Concerning the policy systems for capability re-initialization, capability allocation adaptation and dynamic policies, these policy systems depend on the adaptability nature of service systems. To make generic policy systems, generic behaviors representing the adaptability nature of any service system must be defined and formalized.

Ontology-based capability selection mechanism

Ontology-based capability selection mechanism is a mechanism for exploring the relationship of capabilities and determining the equivalence of capabilities based on a defined capability ontology model. The ontology-based capability selection mechanism has been dealt with in Paper A.

Dynamic policies

Dynamic policies mean the policies are not fixed while the fixed policies are denoted as static policies. Dynamic policies can be changed based on the consequences when the policies are used, which will be calculated from certain goodness criteria. Feedbacks as well as the income and cost functions of the application service systems can be used as the goodness criteria. Income functions indicate the present income of the service provider. Cost functions indicate the present expense of the service provider. The formalized model and mechanism of the dynamic policies can be found in Paper E, Paper F and Paper G.

Parameter production

The parameters *system constraints* and the *reasoning conditions* of RM-based role figures can be reproduced based on *changes* in the environment, as defined in the definition of adaptable service systems, and goodness criteria. The formalized model and mechanism of the parameter production can be found in Paper H.

C3: Capability configuration management

This contribution is about the capability configuration management in the TAPAS service functionality architecture. The capability configuration management comprises i) *capability initialization and re-initialization* and ii) *capability allocation adaptation*.

Capability initialization

The capability initialization is the allocation of the capabilities for the EFSM-based service components to be distributed and instantiated. The configuration of the capability initialization is autonomously generated by an RM-based role figure. The capability initialization configuration consists of EFSM service component types and the targeted actors to instantiate these EFSM service component types. The instantiated EFSM service component types are denoted as EFSM service component instances.

To generate the capability initialization configuration, the required capabilities and capability performances of EFSM service component types are matched with the offered capabilities and capability performance as illustrated in Figure 13. The capability initialization has been dealt with in Paper B and Paper C.

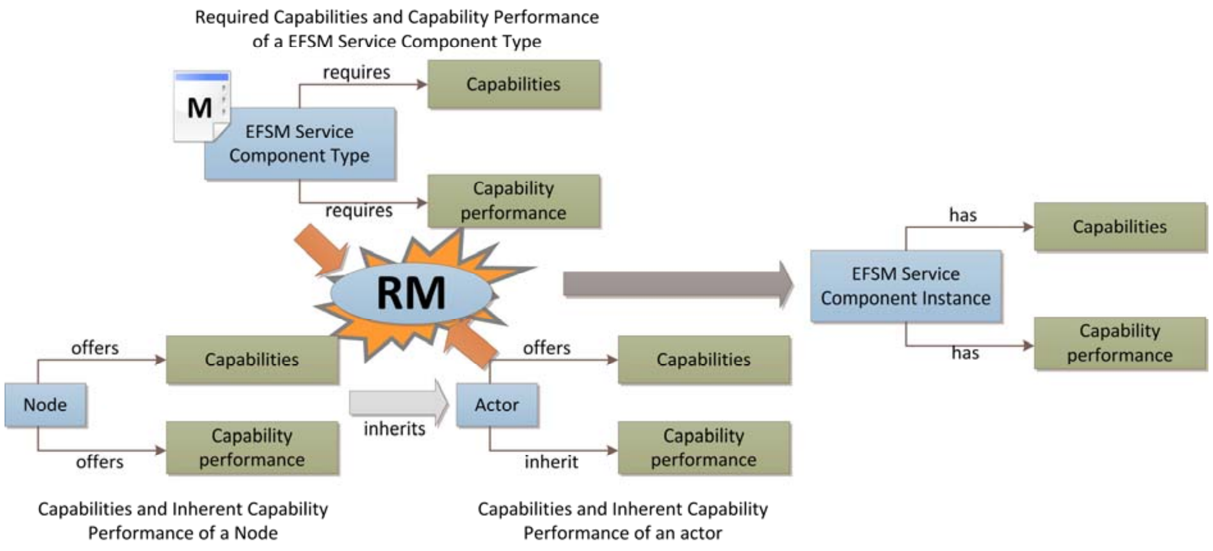

Required Capabilities and Capability Performance of a EFSM Service Component Type
M
EFSM Service Component Type
requires
Capabilities
requires
Capability performance
RM
has
Capabilities
EFSM Service Component Instance
has
Capability performance
offers
Capabilities
Node
offers
Capability performance
inherits
offers
Capabilities
Actor
inherit
Capability performance
Capabilities and Inherent Capability Performance of a Node
Capabilities and Inherent Capability Performance of an actor

Figure 13 – Capability initialization

Capability re-initialization

The capability re-initialization is the re-distribution and re-instantiation of service components and capabilities during the situations when the instantiated service systems are unable to adapt satisfactorily as well as when capabilities are moved or relocated. In case of failures, capability re-initialization finds equivalent capabilities to replace the failed ones based on the ontology-based capability selection mechanism. With the equivalent capabilities, the service system can still offer services with the same level of performance or slightly degraded performance. The capability re-initialization has been dealt with in Paper B and Paper D.

Capability allocation adaptation

The capability allocation adaptation is the monitoring of the performance of the executing service system and the reallocation of capabilities within the executing service systems. A capability allocation adaptation system is formed by a feedback loop as illustrated in Figure 14. The system has policies and parameters. The parameters are system constraints and reasoning conditions. The RM-based *capability manager* $\mathcal{R}_{CM}$, gives suggested actions to an invoking EFSM-based *capability manager* E_{CM} based on the policies and parameters. A dedicated EFSM E_{Σ} activates and de-activates the E_{CM} based on the reasoning procedure.

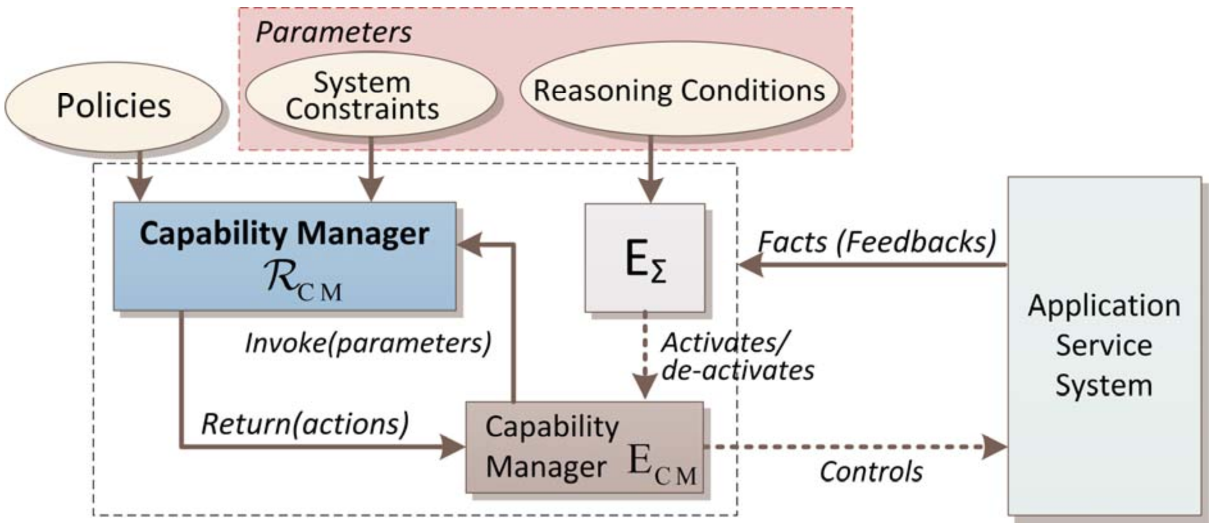


Figure 14 - Capability allocation adaptation

The policies and parameters of the capability initialization, re-initialization and allocation adaptation can be changed. The changes of policies and parameters of these policy-based adaptation models are based on the dynamic policies and the parameter production respectively. An additional feedback loop is used to evaluate the consequences of the used policies and parameters. In the solution framework, the dynamic policies and the parameter production have been used with the capability allocation adaptation. However, it is possible use the parameter production with the capability initialization and re-initialization as well. Note that this PhD study does not intend to obtain the optimized policies and parameters for the capability allocation adaptation. The dynamic policies and parameter production, nevertheless, serve as a flexible tool for the experimentation with alternative policies with respect to optimization.

Figure 15 illustrates the dynamic policies for capability allocation adaptation. An additional feedback loop is formed by an EFSM-based and an RM-based *policy composer*. The policies of the capability allocation adaptation model will be composed. Transformation constraints used by the policy composer consist of rules and actions, which will constitute the policies of the capability allocation adaptation model. The policy composer uses transformation policies to compose and give the policies of the capability allocation adaptation model as the output. The facts are the inherent capability and service performance measures, which are feedbacks from the service system being controlled by capability allocation adaptation.

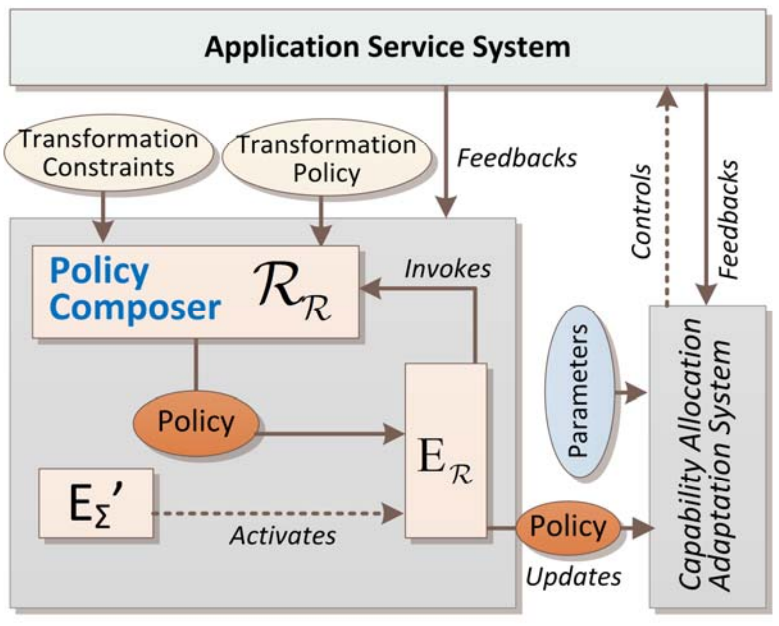


Figure 15 – Dynamic policies for capability allocation adaptation

Figure 16 illustrates the parameter production for capability allocation adaptation. An additional feedback loop is formed by an EFSM-based and an RM-based *parameter producer*. The system constraints and the reasoning conditions will be produced. The parameter producer takes an SLA as the input and produces a system performance requirement model and a physical performance requirement model as the output. Then, the parameter producer takes the produced system performance requirement model and a physical performance requirement model as the inputs and produces the system constraints and the reasoning conditions as the outputs. The transformation is based on the transformation policies. The facts are the inherent capability and service performance measures, which are feedbacks from the service system being controlled by capability allocation adaptation.

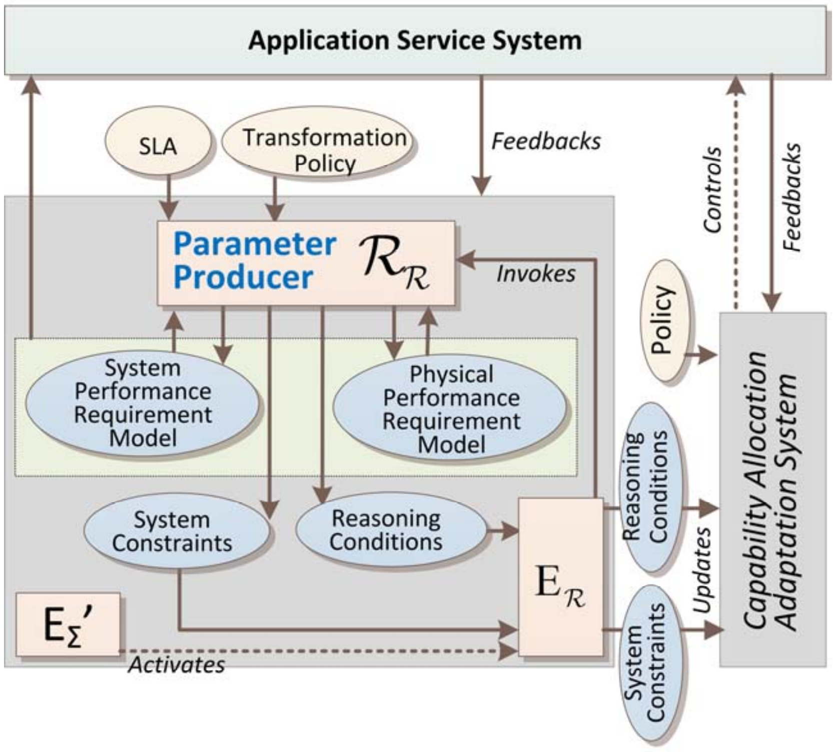


Figure 16 – Parameter production for capability allocation adaptation

The capability allocation adaptation can be found in Paper E, Paper F, Paper G and Paper H. The dynamic policies for the capability allocation adaptation can be found in Paper E, Paper F and Paper G. The parameter production for the capability allocation adaptation can be found in Paper H.

C4: Concept model and data representation

A concept model and a data representation are applied for modeling and formalizing necessary concepts. This contribution consists of the concept ontology and the unified XML-based data representation sub-contributions.

Concept ontology

The concept ontology describes the ontology model for the concepts, which are capability, capability performance, service performance, SLA, EFSM and RM. Concerning the capability, capability performance and service performance, there are various representation standards. Examples are Common Information Model (CIM) [Dmt07a], Simple Network Management Protocol - Management Information Base (SNMP-MIBs) and Management Information Format (MIF). CIM is a conceptual information model for modeling capabilities regardless of the implementation model. SNMP-MIBs are databases for storing information related to capabilities of targeted nodes. MIF is a format used by Desktop Management Interface to store capabilities of desktop or notebook node types. However, the concept ontology is based on CIM because CIM is independent of implementation. It is also possible to map CIM models to SNMP-MIBs as well as MIFs if needed.

The direct use of CIM does not give features that will satisfy all the core functional adaptability requirements. So to model capability requirements, additional modeling language such as UML is needed. UML can provide a set of predefined constructs, such as subClassOf, minCardinality and maxCadinality. However, UML lacks functionality to representing inherent relationships and complex constraints. Moreover, the specification based on CIM and UML needs to be transformed into executable specifications that are recognized by reasoning machines, which may introduce complexity and errors caused from the transformation process. Semantic web languages such as RDF [W3c04a], RDFS [W3c04b] as well as OWL [W3c04c] can also be applied for additional constructs for describing more expressive ontology models of the capability, capability performance, service performance and SLA. The capability, capability performance, service performance ontology has been dealt in Paper A and Paper B.

As defined earlier, SLA is modeled as classes; each of which represents the agreed priority, functionality, performance, payment and penalties functions for a group of users with the same degree of satisfaction and costs.

An EFSM-based specification consists of *EFSM-related data entities* that are a set of states, an initial state, variables, parameters, input signal with parameters, output signal with parameters, state transitions, output functions and the functions and tasks performed during a specific state.

An RM-based specification consists of *RM-related data entities* as illustrated in Figure 17. These data entities are facts, policies, reasoning conditions, constraints and transformation clauses, which have been defined in the generic reasoning model sub-contribution. In this thesis, the use of policies is related to capability configuration management, which consists of capability initialization policies, capability re-initialization policies and capability allocation adaptation policies. Reasoning conditions represent the conditions that activate and de-activate reasoning machines. The capability configuration management conditions are related to the capability allocation adaptation, the capability allocation with dynamic policies and the capability allocation with parameter production. Initial requests such as capability initialization request, capability re-initialization request and capability allocation adaptation request will be formulated as transformation clauses that will be transformed until final transformation clause(s) are derived.

The EFSM-based and RM-based data entities have been dealt with in Paper E, Paper F, Paper G, Paper H and Report I.

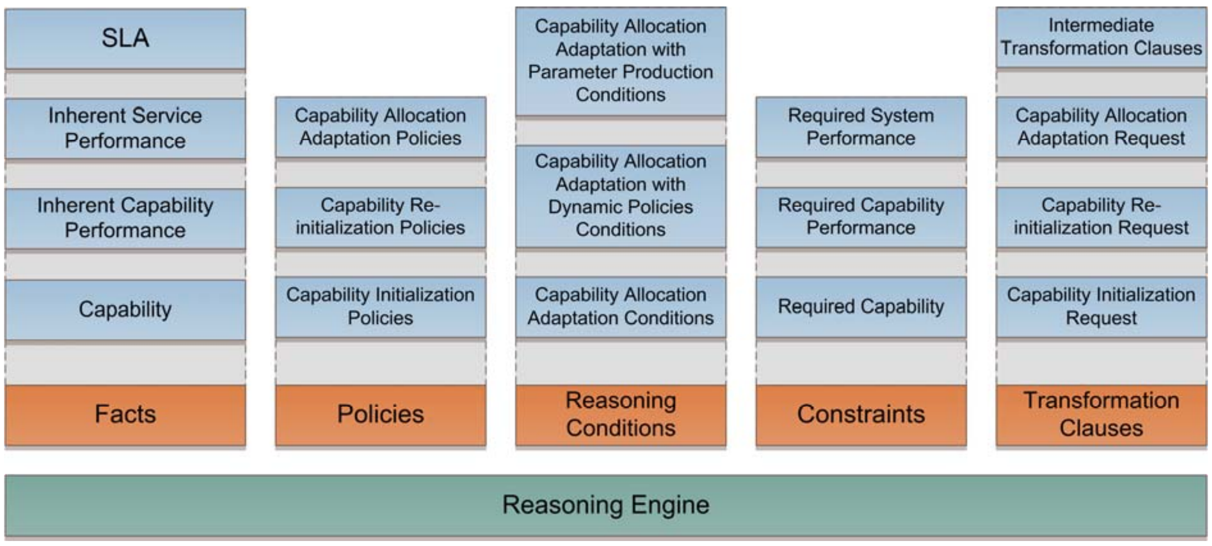

SLA
Inherent Service Performance
Inherent Capability Performance
Capability
Facts
Capability Allocation Adaptation Policies
Capability Re-initialization Policies
Capability Initialization Policies
Policies
Capability Allocation Adaptation with Parameter Production Conditions
Capability Allocation Adaptation with Dynamic Policies Conditions
Capability Allocation Adaptation Conditions
Reasoning Conditions
Required System Performance
Required Capability Performance
Required Capability
Constraints
Intermediate Transformation Clauses
Capability Allocation Adaptation Request
Capability Re-initialization Request
Capability Initialization Request
Transformation Clauses
Reasoning Engine

Figure 17 – The RM-related data entities

Data representations

The data representations of capability, capability performance, service performance are based on xmlCIM [Dmt07b]. The data representation of SLA is RDF, which is chosen because of none of the standards can represent the complete set of ontology attributes that are required to model an SLA class.

The representations of the EFSM- and RM-based specifications are based on XML-based constructers. There are four types of XML-based constructers: ordinary XML, XML expressions, XML clauses and XML rules. An ordinary XML is just ordinary XML elements without variables. XML expressions are XML documents that consist of variables. The XML expressions are expressive and used to model implicit knowledge. An XML clause consists of XML expressions and used to represent certain knowledge during a capability configuration. XML rules are used by reasoning machines to transform an XML clause until the final clause representing the answers is obtained. The facts are modeled by ordinary XML. The reasoning conditions, constraints are modeled by XML expressions. The transformation clauses are modeled by XML clauses. The policies are modeled by XML rules.

The RM-based data representation has been dealt with in Paper A, Paper G, Paper E and Report I.

C5: Scenarios – experimentation and simulation

Five scenarios (S1-S5) have been designed for the evaluation and validation of the other contributions. The scenarios are adaptable service system applications that are created based on the concepts, models and mechanisms of the solution framework. These scenarios are as follows:

S1: Ontology-based capability selection for a Tele-school application

Ontology-based capability reasoning for a Tele-school application is presented in Paper A. The sound capability of the clients in the Tele-school application is considered. By the use of RDFS's *subClassOf*, the sound capability and the loudspeaker capability are defined as substitutable. Based on the demonstration method, the reasoning result shows that the mobile clients having capabilities that are sub-class of the sound capability must switch off the capabilities when entering the classroom.

S2: Printing system capability initialization and re-initialization

Capability initialization and re-initialization of Intelligent Printing Management (IPM) service system has been demonstrated in Paper B and Paper C. The scenario illustrates the autonomous generation of capability configuration and re-configuration plans of the roles of IPM. Each role has different requirements. The proposed capability initialization and re-initialization policies determine the appropriate node to instantiate each role.

S3: Capability re-initialization of RM-based role figures

Capability re-initialization of RM-based role figures is given in Paper D. The scenario presents the ability of RM-based role figures to re-initialize its capabilities when there are failures in the service system. In the scenario, database clients connect to the database server to access the data. The database clients may need to re-initialize its capabilities when security treats are detected and the database server blocks the connections. The database clients use policies to determine the appropriate action to perform; e.g. moving to a new node. This scenario is based on the demonstration method.

S4: Policy-based capability allocation adaptation for streaming service

Policy-based streaming service capability allocation adaptation simulation is presented in Paper E and Paper F. The scenario defines a capability allocation adaptation model for an on-demand music video streaming service system. The capability model can have static and dynamic policies. The policy system of the capability allocation adaptation model is composed based on goodness criteria. The performance of the service system under the capability allocation adaptation is compared by the situations when the capability allocation adaptation uses static policies, dynamic policies and no policy (no capability allocation adaptation).

S5: Policy-based parameter production for policy-based capability allocation adaptation

This is an extension the simulation-based scenario: Policy-based capability allocation adaptation. The scenario is presented in Paper H. The scenario defines a capability allocation adaptation model for an on-demand music video streaming service system. The parameters of the capability allocation adaptation are produced based on inherent capability and service performance and the cost function of the system. The performance of the service system under the capability allocation adaptation is compared by the situations where the capability allocation adaptation has static parameters and when it has dynamic parameters.

3.2 The realization of the problem statements

Figure 18 illustrates the relationship between the problem statements **P1-P6**, and the contributions of the solution framework as well as the relationship between the solution framework and the publications in **Part II**. The sub-contributions that are orange boxes have been applied in the work reported in all papers.

Considering **P1**, the capability-based computing architecture (Figure 9) is applied. The RM-based functionality is intended to meet the general properties. The capability concept is needed to meet the core properties **A1**, **A2** and **A3**. The capability, service performance and SLA concepts, however, are the concepts needed to meet the core properties **A2** and **A3**. The use of the capability, capability performance, service performance and RM-based specification concepts can be found in all papers. The concept SLA, however, is used in Paper E, Paper F, Paper G and Paper H for the provisioning of QoS in adaptable service systems.

Concerning **P2**, The capability configuration management is applied in the service architecture as illustrated in Figure 4 for capability management functionality modeling. The capability initialization is related to the core property **A1**. The capability re-initialization and allocation adaptation are related to the core properties **A2** and **A3**. The capability initialization is demonstrated in Paper B and Paper C. The capability re-initialization is demonstrated in Paper B and Paper D. The capability allocation adaptation is dealt with in Paper E, Paper F, Paper G and Paper H. The formalized models of capability initialization, re-initialization and capability allocation adaptation models are in Paper G.

Concerning **P3**, autonomic capability management requires the capability configuration management to generate capability initialization, re-initialization and allocation adaptation configurations dynamically. Based on the generic reasoning model, the capability configuration management mechanism is activated and de-activated autonomously. The application service systems are monitored. Changes in the application service systems as well as the environment will trigger a feedback loop constituting the capability allocation adaptation mechanism to manage the behavior of the service systems. Important settings for the capability allocation adaptation are policies and parameters. The policies of the capability allocation adaptation mechanism can be either static policies or dynamic policies. The parameters of the capability allocation adaptation can also be produced by the parameter production. The capability initialization can be found in Paper B and C. The capability re-initialization can be found in Paper B and D. The capability allocation adaptation and the dynamic policies can be found in Paper E, Paper F and Paper G. The parameter production for the capability allocation adaptation can be found in Paper H. The software implementation of the generic reasoning model can be found in Report I.

Concerning **P4**, the ontology to be used has been defined as the concept ontology: capability, capability performance, service performance, SLA, EFSM and RM in the capability-based computing architecture. The capability, capability performance, service performance and SLA ontology are needed to be stored and made available. These ontology instances are included in the EFSM-related and RM-related data entities. Standard representation models – Common Information Model (CIM) and Resource Definition Framework (RDF) are applied to model the concepts: capability, capability performance, service performance and SLA as presented in Paper A and Paper B. The RM-based data representation for modeling the RM-related data entities have been formalized in Paper E, Paper F, Paper G and Paper H.

Concerning **P5**, The policy-based reasoning is required to model capability configuration management functionality with respects to modularity, reusability, expressive power and flexibility. The policy-based specification is flexible, expressive and powerful. RM-based role figures have ability to download and execute policies that can be stored, managed and easily distributed. The generic policy system definition allows policies to be composed in a modular and reusable away. The ontology-based capability selection mechanism gives expressive reasoning power to the capability configuration management to reason based on a defined capability ontology model. The RM-based role figures have been formalized and used in Paper E, Paper F, Paper G and Paper H. The generic policy system definition has been demonstrated in Paper C. The ontology-based capability selection mechanism has been demonstrated in Paper A.

The dynamic policies and parameter productions are techniques that can be used to evaluate the usability and goodness of the policies and parameters of a capability configuration management system. The goodness of the used policies and parameters are observed based on feedbacks, income functions and cost functions. The dynamic policies have been formalized, demonstrated, evaluated and used together with the capability allocation adaptation in Paper E, Paper F and Paper G. The parameter production has been formalized, demonstrated, evaluated and used with the capability allocation adaptation in Paper H.

Concerning **P6**, the solution framework is evaluated and validated by demonstration and simulation. The five scenarios defined by the contribution C5 are: S1: Ontology-based capability selection for a Tele-school application, S2: Printing system capability initialization and re-initialization, S3: Capability re-initialization of RM-based role figurers, S4: Policy-based capability allocation adaptation for streaming service, and S5: Policy-based parameter production for policy-based capability allocation adaptation

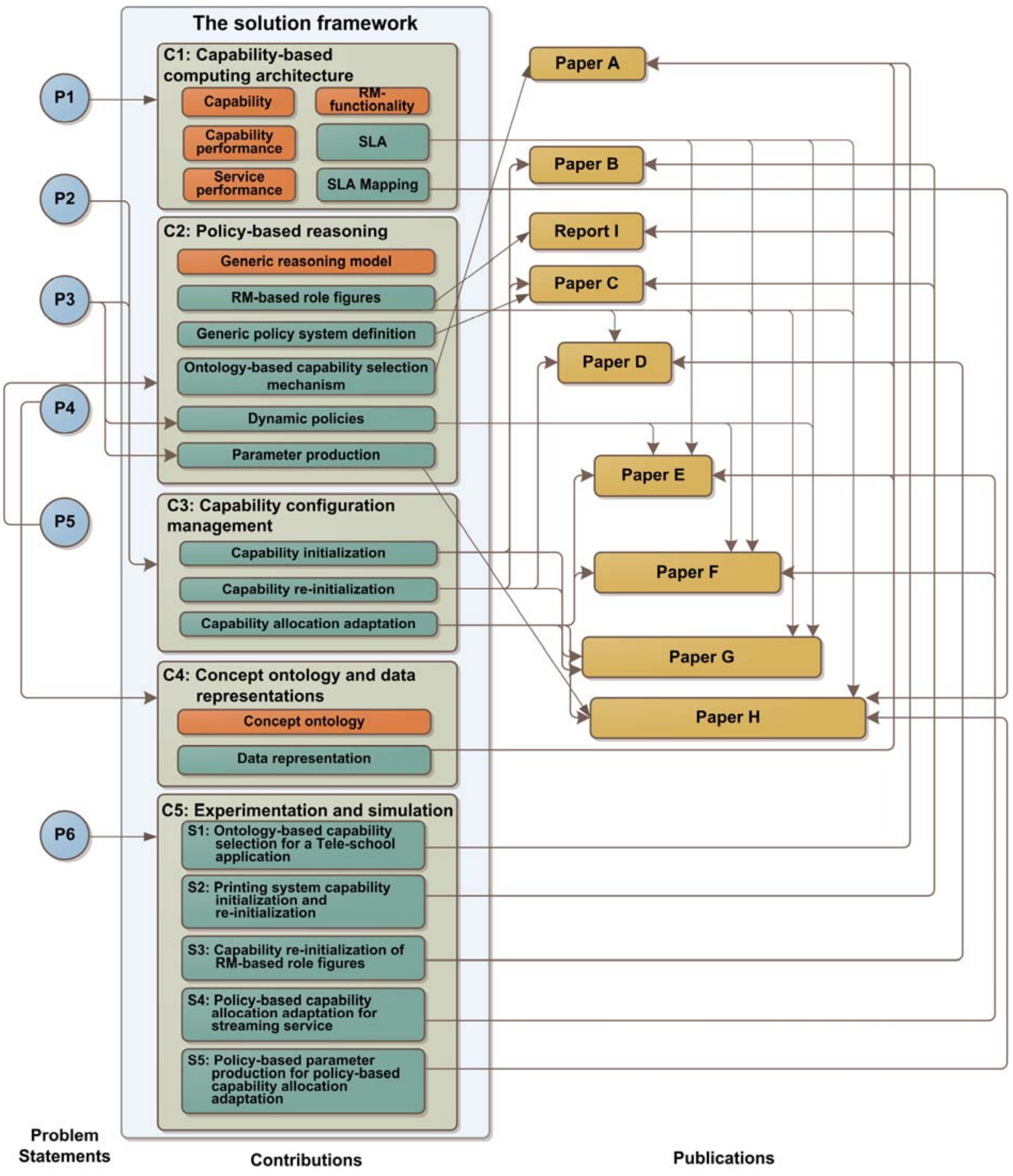

The solution framework
C1: Capability-based computing architecture
Capability
RM-functionality
Capability performance
SLA
Service performance
SLA Mapping
C2: Policy-based reasoning
Generic reasoning model
RM-based role figures
Generic policy system definition
Ontology-based capability selection mechanism
Dynamic policies
Parameter production
C3: Capability configuration management
Capability initialization
Capability re-initialization
Capability allocation adaptation
C4: Concept ontology and data representations
Concept ontology
Data representation
C5: Experimentation and simulation
S1: Ontology-based capability selection for a Tele-school application
S2: Printing system capability initialization and re-initialization
S3: Capability re-initialization of RM-based role figures
S4: Policy-based capability allocation adaptation for streaming service
S5: Policy-based parameter production for policy-based capability allocation adaptation
P1
P2
P3
P4
P5
P6
Paper A
Paper B
Report I
Paper C
Paper D
Paper E
Paper F
Paper G
Paper H
Problem Statements
Contributions
Publications

Figure 18 – Problem statements, contributions and publications

4. Description of the publications in Part II

4.1 Paper A - An Approach to Capability and Status Modeling

Paramai Supadulchai and Finn Arve Aagesen

In Proceedings of *Norwegian Informatics Conference (NIK 2003)*,
Stavanger, Norway, 2004

Abstract

A recent trend in network systems is the advanced technology to dynamically handle changes in the system. An important basis relies on the integration of capability and status representation and their semantic descriptions, which is currently not expressive enough. In this paper, we propose a Unified Capability and Status Representation Framework (UniCS) for handling several aspects related to capability and status. A scenario is modeled by UniCS to show how a network system exhibits adaptable behavior.

Contributions

C2{Ontology-based capability selection mechanism}, **C4**{Concept ontology, RM-based data representation}, **C5** {Ontology-based capability selection for a Tele-school application}

Summary

In this Paper a *Unified Capability and Status Representation Framework* (UniCS)[1] is presented. UniCS is a *formal representation* and a *computation framework* for the capability configuration management. The representation is classified in three aspects as: *1) the* syntactic aspect, *2) the semantic* aspect and 3) the computation *aspect*.

In the *syntactic aspect*, physical capabilities can be represented in any standard representation. UniCS allows use of multiple physical capability and

[1] UniCS has been extended as the present data model

service performance measure representations such as CIM, UPnP, RDF or any other proprietary XML-based representations.

In the *semantic aspect*, the framework encourages the use of semantic descriptions such as *ontology* to formally describe the semantic of certain capability, capability performance and service performance. In addition, an ontology-based *play view capability dimension* is suggested for simplicity, unified representation and domain-oriented semantic. The play view capability is aimed for:

- *Simplicity* – Unnecessary details of the physical capabilities are abstracted away; only focused capability properties are considered.
- *Unified representation* – The capability configuration specification is needed to be written only once independent of physical capability representation used.
- *Domain-oriented semantic* – Capabilities are given specific semantic in a certain domain.

In the *computation aspect*, a reasoning functionality can be used as follows:

- *Transform physical capabilities to play-view capabilities* – Capabilities represented by multiple representations will be transformed into a *unified* play-view capability representation.
- *Ontology reasoning* – the reasoning mechanism can reason about capability and service performance measure relationships based on used ontology's vocabularies.

The paper's main contribution is the concept model and data representation, where the capability, capability performance and system performance concepts are defined, formalized and represented in the proposed XML-based data representation. The paper also has a contribution to the ontology-based capability selection mechanism. Scenario S1 – ontology-based capability selection for a Tele-school application is presented to illustrate how this ontology-based capability selection mechanism is possible in UniCS. Two capabilities are defined as substitutable by the use of RDFS's *subClassOf*. The reasoning result shows that these two capabilities can be seen equivalent by the service system.

While the use of UniCS adds more simplicity and semantic to the capability configuration management, a drawback is that the capabilities must always be converted into the unified play view capability format. This requires additional transformation rules to be written.

4.2 Paper B – Configuration Management for Adaptable Service Systems

Finn Arve Aagesen, Paramai Supadulchai, Chutiporn Anutariya and
Mazen Malek Shiaa

In Proceedings of *IFIP International Conference on Metropolitan Area Network, Architecture, Protocols, Control and Management (MAN 2005),*
Ho Chi Minh City, Vietnam, 2005

Abstract

Adaptable service systems are service systems that are capable of handling dynamic changes in both time and position related to users, capabilities, nodes and changed service requirements. The paper presents a formal framework for dynamic configuration and reconfiguration of services in TAPAS (Telematics Architecture for Play-based Adaptable Systems). The framework presented in this paper, provides representation and reasoning mechanisms for semantic description and matching of required and offered capabilities and status which are required by a particular service system. It employs CIM and RDF based on XML as well as the XML Declarative Description Language (XDD) to provide human readable and machine-comprehensible descriptions of status, capabilities, system (re)configuration plans as well as the exchange of messages. A reasoning system for Configuration Management has been developed by use of XET (XML Equivalent Transform).

This system can directly operate and reason about XML elements and XML clauses described by XDD. The system is demonstrated for a simple Intelligent Printing Management System.

Contributions

C3{Capability initialization, Capability re-initialization}, **C5** {Printing system capability initialization and re-initialization}

Summary

The paper presents *capability initialization* and *capability re-initialization* of the capability configuration management for adaptable service systems. The paper focuses on three aspects:

- TAPAS architecture
- Dynamic configuration framework for capability initialization and re-initialization
- Data representation

TAPAS architecture is separated into a computing architecture and a system management architecture. The computing architecture is a generic architecture for modeling any service systems and their components. The system management architecture is the structure of the system management components. In the computing architecture, a service consists of service components that are realizing by roles and deploy on physical nodes.

The capability configuration and re-configuration determine appropriate nodes to deploy roles and instantiate role figures. This is accomplished by a *dynamic configuration framework* constituting *configuration management functionality* in the system management architecture. Dynamic means that the capability configuration is not fixed and can be determined based upon available capability and service performance measures of nodes. An XET-based reasoning machine works as a *capability configuration manager* that transforms a service request, a service component request or a trouble report into a configuration plan for a *service manager* to deploy the needed roles.

Scenario S2 – printing system capability initialization and re-initialization describes the composition of dynamic capability configurations for an *Intelligent Printing Management* (IPM) service system. The required capability and service performance measures of each role are defined in *play (re-)configuration rules* used by the configuration manager. Based upon available capability and service performance measures of nodes, roles can be instantiated at the nodes with the *sufficient* capabilities and service performance measures.

4.3 Paper C – A Framework for Dynamic Service Composition

Paramai Supadulchai and Finn Arve Aagesen

in Proceedings of *1st International IEEE Workshop on Autonomic Communication and Computing (ACC 2005)*, Taormina, Italy, 2005.

Abstract

To be able to utilize the generative potential of future networks for service composition, the attributes of services and networks must be appropriately formalized, stored and made available. Important attributes are the capability and the status. A capability is an inherent property of a node or a user, which defines the ability to do something. A capability in a network node is a feature available to implement services. A capability of a user is a feature that makes the user capable of using services. Status is a measure for the situation in a system. This paper proposes a representation framework for capability and status, denoted as Unified Capability and Status Representation Framework (UniCS). This framework is used to decide upon dynamic use of capabilities, and is used to support the dynamic composition of a service system. UniCS consists of facts and configuration rules. The facts describe the availability and requirement of capabilities and status of a service system. The configuration rules verify, manipulate, transform and discover new facts with defined axioms and constraints. An instance of UniCS is the input specification for a reasoning engine to dynamically generate a composition plan for a service system.

Contributions

C2{Generic policy definition}, **C3**{Capability initialization}, **C5** {Printing system capability initialization and re-initialization}

Summary

A dynamic service composition framework is presented in Paper C. This paper extends the capability initialization in Paper B by making a generic policy system definition for capability initialization, which is the main achievement of this paper. The demonstration scenario is this paper is scenario S2 – printing system capability initialization and re-initialization.

In this paper, a service system is realized by *play sessions* that are formal specifications of the collaboration between roles. A role also has capability requirements denoted as *role requirements*.

Unlike Paper B, The dynamic capability initialization in this paper is based on UniCS. The paper demonstrates the use of Web Ontology Language for Semantic Web (OWL-S) to describe the semantics of a play consisting of play sessions.

4.4 Paper D - Autonomic Service Configuration by a Combined State Machine and Reasoning Engine-based Actor

Paramai Supadulchai and Finn Arve Aagesen

In Proceedings of *the 2005 IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM 2005),*
Delta Centre-Ville Hotel, Montréal, Canada, 2005

Abstract

Service systems constituted by service components are considered. Service components are executed as software components in nodes, which are physical processing units such as servers, routers, switches and user terminals. A capability is an inherent property of a node or a user, which defines the ability to do something. Status is a measure for the situation in a system. A service system has defined requirements to capabilities and status. Because of continuous changes in capabilities and status, dynamic service configuration with respect to capabilities and status is needed. Software components are generic components, denoted as actors. An actor is able to download, execute and move functionality, denoted as a role. Configuration is based on the matching between required capability and status of a role and the present executing capabilities and status of nodes. We propose an approach for role specification and execution based on a combination an Extended Finite State Machine and a rule based reasoning engine. Actor execution support consisting of a state machine interpreter and a reasoning engine has been implemented, and has also been applied for a service configuration example.

Contributions

C2{RM-based role figures}, **C3**{Capability re-initialization}, **C4**{RM-based data representation}, **C5**{Capability re-initialization of RM-based role figures}

Summary

In this paper, a new *actor model* used in the TAPAS computing architecture has been proposed. Service systems consisting of service components are executed as software components, which are denoted as actors. The classical EFSM-based actor model is supplemented with a Reasoning Machine (RM)-based functionality that makes rule-based decisions possible. An actor playing a role as denoted as role figures can execute both EFSM-based specification and RM-based specification.

The use of the actor's RM-based functionality for a service configuration enables actors to take decision by themselves. This eliminates the need to have a centralized service configuration manager. The service configuration is based on role requirements and inherent capabilities and service performance. The role requirement describes required capabilities to execute a certain role. An actor can determine based upon its assigned role and the capabilities information of nodes what to do. For example, an actor may choose to move if the present node it stays does not have enough required capabilities.

The XML-based data representation for the RM-based specification is presented. However, this paper does not have formalized RM-related data entities.

Scenario S3 – capability re-initialization of RM-based role figures demonstrates a distributed capability re-initialization. An executing actor, when detects a security threat, can determine a node that is possibly safe and has enough required capabilities in order to execute its role.

4.5 Paper E - Policy-based Adaptable Service Systems Architecture

Paramai Supadulchai and Finn Arve Aagesen

In Proceedings of *the IEEE 21st International Conference on Advanced Information Networking and Applications (AINA-07)*,
Niagara Falls, Canada, 2007

Abstract

This paper presents a policy-based architecture for adaptable service systems based on the combination of Reasoning Machines and Extended Finite State Machines. Policies are introduced to obtain flexibility with respect to specification and execution of adaptable service systems that give high performance over a range of system status values. The presented architecture covers three aspects: service system framework, adaptation mechanisms and data model. The adaptation mechanisms can be based on static or dynamic policy systems. Static policy systems have a non-changeable set of policies, Dynamic policy systems have a changeable set of policies, which are managed by policies at a higher level. The data model for the reasoning machine functionality is based on the rule-based reasoning language "XML Equivalent Transformation" (XET). The capability configuration management of a service system with runtime simulation results based on the proposed architecture is presented with the intention to illustrate the use of the architecture and discuss the potential advantages of using dynamic policies.

Contributions

C1{SLA}, **C2**{RM-based role figures, Dynamic policies}, **C3**{Capability allocation adaptation}, **C4**{EFSM and RM-related data entities, RM-based data representation}, **C5** {Policy-based capability allocation adaptation for streaming service}

Summary

The paper presents an architecture for policy-based adaptable service system. Aspects covered comprise

A) Service system framework,

B) Adaptation mechanisms and

C) Concept model and data representation.

The service system framework aspect is a general component structure, the integration of a proposed Reasoning Machine (RM) functionality into the component structure, application domains of the RM functionality and a reasoning procedure used by the RM functionality. The adaptation mechanism aspect concerns the use of *appropriate* policies to control service systems when the reasoning functionality is needed. The concept model and data representation aspect concerns all the data representation related to the RM functionality.

In the service system framework's general component structure, a service system consists of executing service components, each of which belongs to a certain service component type. A service component type can be a pure EFSM or a combination between EFSM and RM functionality, which is more flexible and expressive for a decision making role. EFSM may have requirements with respect to capabilities, capability performance and service performance. In the paper, a formalism of the RM functionality is given. The potential applications of the RM functionality are as follows.

1. The RM functionality can be used as a traditional procedural service for an EFSM-based service component to take decisions, which are not involving the management of capabilities and system performance.
2. The RM functionality can be used for the initial configuration and reconfiguration of service components according to EFSM's capability, capability performance and service performance requirements.
3. The RM functionality can be used for the allocation adaptation of the service system behavior based on the system's capability performance and service performance.
4. The adaptation of policies used in 1-3.

These applications are based on the reasoning procedure, which is a procedure used by an RM to select rules to be applied.

A policy system consists of a set of *rules* and *actions*. The policy-based adaptation mechanism is based on a feedback loop. An RM receives feedbacks, which are capability and service performance measures from an application service system and uses policies to suggest actions that will be used to manage the behavior of the application service system when it is entering a reasoning condition. The policies used are of two types: static policies and dynamic policies. The static policies are non-changeable. Dynamic policies mean the policy rules can be changed. The functionality is based on some goodness criteria, reference inputs as well as feedback from the application service system, which will used to evaluate policy rules and subsequently change the rules' priority, suspense them for a certain time period or completely remove them from the policy system.

The data representation for the RM-based specification is based on XML Equivalent Transformation (XET), which can represent ordinary XML documents as well as XML expressions and XML rules. Ordinary XML documents are used to represent capability, capability performance measures and system performance measures. XML expressions are ordinary XML documents with XML variables. They are used to represent system constraints of the RM functionality. XML rules represent the specification of RM-based applications in 1-4 above.

Scenario S4 – policy-based capability allocation adaptation for streaming is presented. The intention is to illustrate the use of the proposed architecture and the potential advantage of using dynamic policies. Users belong to classes. The QoS of each class is described by SLA. The users of the class with higher priority have more privilege in using the service and allocating capabilities. This mechanism is based on the capability allocation adaptation.

Regarding the service system, the scenarios include the uses of no policy, the capability allocation adaptation with static policies and the capability allocation adaptation with dynamic policies. The presented rules, however, are not the most optimum case. In fact, it is assumed that when the service system is operated under dynamic policies, it will be able to learn the consequences of the rules and remove some rules that are not appropriate in the current situation.

There are situations where the use of no policy can be superior or equal to the use of policies. This can happen when the service system is operated under some optimized system parameters. However, the same set of system parameters will likely not be optimal for other system traffic load cases. In the presented scenarios the use of no policy and one server is a good solution in the low traffic case, while the use of no policy and two servers is a good solution in the high traffic case. In the given scenarios, the service system operated under static policies give a relatively high income in both low and high traffic. The service system operated under dynamic policies, however, has a performance which is superior or equal to other scenarios in both the low and the high traffic case. When the service system operated under dynamic policies, the rules that are not appropriate were suspended for a certain time period as expected.

In addition to having the potential for providing optimal solutions covering dynamic traffic situations, the proposed architecture also is a flexible tool for the experimentation with alternative policies with respect to optimization.

4.6 Paper F - Towards Policy-Supported Adaptable Service Systems

Paramai Supadulchai and Finn Arve Aagesen

In Proceedings of *the 13th Eunice Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services*, University of Twente, the Netherlands, 2007

Abstract

This paper presents a policy-supported architecture for adaptable ser-vice systems based on the combination of Reasoning Machines and Extended Finite State Machines. Policies are introduced to obtain flexibility with respect to specification and execution of adaptation mechanisms. The presented architecture covers two aspects: service system framework and adaptation mechanisms. The service system framework is a general framework for capability management. Adaptation mechanisms are needed for autonomous adaptation. The adaptation mechanisms can be based on static or dynamic policy systems. Capability management for of a simple music video-on demand service system with runtime simulation results based on the proposed architecture is presented.

Contributions

C1{SLA}, **C2**{RM-based role figures, Dynamic policies}, **C3**{Capability allocation adaptation}, **C4**{EFSM and RM-related data entities}, **C5** { Policy-based capability allocation adaptation for streaming service}

Summary

Paper F has some overlapped content with Paper E. However, it provides more simulation results. The paper extends the simulations of the scenario S4 – policy-based capability allocation adaptation for streaming service in Paper E. The simulation in Paper E concerns only high and low traffic load. The simulation in Paper F includes a medium load case and the case that the traffic load varies as a function of time. In the latter scenario, the time when the traffic load is at a fixed level is denoted as a ρ period.

The simulation confirms that there are situations where the use of no policy can be superior or equal to the use of policies, which will when the service system is operated under some optimized system parameters. However, the same set of system parameters will likely not be optimal for other system traffic load cases especially when the traffic load varies as a function of time. In the case where the traffic load varies as a function of time and the system is operated with dynamic policies, it produces a better result.

In this paper, further simulations were conducted to compare the results of the static policies and the dynamic policies for different ρ periods. The result shows that when the value of the ρ period is small, the static policy case may give a better result because the dynamic policy case need more time to learn the consequences of the rules being applied. When the value of the ρ period is increased, the dynamic policy case, however, gives more superior results. The system operated under dynamic policies needs a certain period of time denoted as *learning time* to learn the consequences of policies in order to provide superior performance.

4.7 Paper G - A Capability-based Service Framework for Adaptable Service Systems

Finn Arve Aagesen and Paramai Supadulchai

In Proceedings of *The 2nd International Conference on Advances in Information Technology (IAIT2007)*,
Bangkok, Thailand, 2007

Abstract

This paper presents a capability-based service framework for adaptable service systems. The paper has focus on: I) Adaptability properties, II) The architecture solution needed to meet these properties, and III) Capability configuration management functionality. Adaptability is based on flexibility. Accordingly, basic rearrangement flexibility must be provided. But the framework must in addition have necessary concepts, features and functionality that make it possible to adapt to various traffic situations and failure states. The architecture solution has a computing and a service functionality dimension. The computing architecture is based on a theatre metaphor, where actor, manuscript, role figure and capability are core concepts. Fundamental QoS concepts are capability, capability performance, service performance and service level agreements. Actors, which are the basis for the implementation of the service functionality, can be Extended Finite State Machines or Reasoning Machines. Actors are able to download and execute EFSM- and RM-based manuscripts. Models for capability configuration management functionality are presented. Reasoning machines are used for capability configuration management in general as well as for policy adaptation.

Contributions

C1{Capability, Capability performance, Service performance, SLA, RM-based functionality}, **C2**{RM-based role figures, Dynamic policies}, **C3**{Capability Initialization, Capability re-initialization, Capability allocation adaptation}, **C4**{ EFSM and RM-related data entities}

Summary

Paper G provides a summary of the work in Paper A, B, C, D, E and F. The paper focuses on I) adaptability properties II) the architectural solution needed to meet these properties and III) capability configuration management functionality.

The adaptability properties include the core properties **A1**, **A2** and **A3**, which has been mentioned. The architectural solution defines the necessary features and QoS related concepts for the capability configuration management in adaptable service systems. Such QoS related concepts are capability, capability performance, service performance and service level agreement.

In the architectural solution, the architecture for adaptable service systems is split into two architectures: service architecture and computing architecture. The service architecture represents the functionality of service systems independent of implementation. The computing architecture is related to the specification and execution of service systems independent of functionality set. The service architecture will need a rich set of features to fulfill the adaptability properties **A1-A3**. However, this paper focuses only the *capability configuration management* functionality.

The capability-based computing architecture is founded on a theatre metaphor where play, actor, role, role figure, manuscript, dialogue concepts are defined. The capability-based computing architecture has three viewpoints: service, play and physical view. The service view represents service system, service component and service performance concepts. The play view, however, has the theatre metaphor concepts play, role, role figure and dialogue except the actor and manuscript concept. The physical view has the actor, node, manuscript, capability and capability performance concepts.

The capability-based computing architecture concepts and functionality are intended as a basis for designing functionality that meet the general and core properties **A1-A3**. The play view concepts are primarily rearrangement flexibility oriented **A1**. The QoS concepts, give a basis for functionality that can meet the robustness and survivability as well as the QoS awareness and resource control properties **A2-A3**.

RM-based role figures are considered. An actor can be the combination of EFSM and RM and is able to download and execute EFSM and RM-based manuscripts. The EFSM part has requirement with respect to capabilities and service performance to perform their intended functionality. The RM part makes policy-based specification and operation, which are expressive and powerful, possible.

The capability configuration management is the allocation, re-allocation and de-allocation of capabilities and comprises service system capability initialization and re-initialization and capability allocation adaptation. The capability initialization and re-initialization are the allocation of capabilities for the service components to be distributed and instantiated. Capability allocation adaptation is the monitoring of the performance of the executing service system and the reallocation of capabilities within the executing service systems.

The service component models including generic EFSM and RM models are formally presented. For the cases where capabilities are involved, the functionality domain of reasoning machines can be as A) a reasoning service for service system capability initialization and re-initialization, B) a feedback loop for capability allocation adaptation based on capability and system performance, C) a feedback loop for dynamic policies used in B). Reasoning cluster types are introduced as the integration of the RM-based models in A), B) and C) in the service system. Capability configuration models for A)-B) based on the RM formalism are given.

4.8 Paper H – Autonomous Production of Parameters of an Autonomous Capability Allocation Adaptation Model

Paramai Supadulchai and Finn Arve Aagesen

Prepared for a submission

Abstract

Autonomous capability allocation adaptation within a service framework for adaptable service systems is considered. The basis for autonomous capability allocation adaptation is service level agreements as well as service and capability performance. Capability allocation adaptation is based on capability allocation adaptation models. These models rely on fine-tuned parameters that are system constraints and reasoning conditions. This paper presents a framework for autonomous parameter production for capability allocation adaptation models. A parameter production example for capability allocation adaptation for an online music video on-demand service based on the proposed framework is given.

Contributions

C1{ Capability, Capability performance, Service performance, SLA, RM-based specification and execution }, **C2**{RM-based role figures, Parameter production}, **C3**{Capability allocation adaptation}, **C4**{EFSM and RM-related data entities}, **C5** {Policy-based parameter production for policy-based capability allocation adaptation}

Summary

This paper presents an autonomous parameter production of an autonomous capability allocation adaptation model. The presented issues are as follows:

I. TAPAS service framework and service component models

II. Autonomous capability allocation adaptation models

III. Autonomous parameter production of the parameters of the capability allocation adaptation in II, which is the main contribution of this paper. These parameters are system constraints and reasoning conditions.

The TAPAS service framework is the overall structural and behavior framework for the specification and execution of services. The framework is separated into the computing architecture and the service architecture. The service architecture represents the functionality of service systems independent of implementation. The computing architecture is related to the specification and execution of service systems independent of functionality set. The details of the service and computing architecture have been discussed in the summary of Paper G.

The computing architecture is extended with capability performance, service performance and SLA. An SLA is a contract made between the service provider and service users for describing service offering functionality, agreed offered functionality of service components, agreed performance, payment in case of agreed performance, payment in case of reduced performance and penalties when the SLA is broken. The formalism of an SLA of a group of users belonging to the same QoS class is defined.

SLAs are transformed into system performance requirement models and a capability performance requirement model. These two models will be subsequently transformed into the system constraints and reasoning conditions for the capability allocation adaptation. Rules and dimensioning functions for the transformations are formalized by XET. The parameter production is a generic functionality that can produce parameters for any capability allocation adaptation model. The transformation rules are generic. The dimensioning functions, however, must be re-specified.

Scenario S5 - policy-based parameter production for policy-based capability allocation adaptation is presented. The considered service system is a music video on-demand service. The considered capability is the access link, which can be composed from different server types having different capability capacity and cost. The application service system handles capability allocation adaptation for both service user point of view and service provider point of view. On the service user point of view, QoS of a user is controlled by SLA, which has been transformed as the parameters of the capability allocation adaptation. On the service provider point of view, the system tries to optimize the access link composition and usage based on system performance, capability performance, capability cost and the capability capacity required by users.

The system constraints consists of requirements with respect to the waiting time of users in different classes, the access link capability needed and the server type configuration are based on SLA, inherent system performance measures, present income and/or present cost of the system. The reasoning conditions are related to the present total waiting time of the users. These parameters can be produced at run-time.

Two scenarios are presented. The first scenario considers the capability allocation adaptation where the capability allocation adaptation parameters are reproduced at run-time. This scenario is denoted as dynamic parameters. The second scenario describes the capability allocation adaptation, which the parameters are produced only once. Additional scenarios are considered. In the former case, the traffic load can vary as a function of time while the server cost is fixed. In the latter case, the cost of the server nodes can vary as a function of time while the traffic load is fixed. The server cost depends on the cost associated with different node types and the number of server required. When the traffic load is varied, the time when the traffic load is at a fixed level is denoted as a ρ period. When the server cost is varied, the time when the server cost is fixed at a certain level is denoted as a J period.

When the traffic load varies as a function of time, the dynamic parameter case gives relatively better results. The static parameter case seems to require some learning time to fine tune the appropriate number of servers parameter at the beginning of each ρ period while in the dynamic case, this parameter is recalculated. When the server cost varies as a function of time, the dynamic parameter case has relatively less cost due to the ability to switch to cheaper combinations of node types when the cost functions are varied.

The production of capability adaptation model parameters can be a flexible tool for the experimentation with alternative parameters of capability allocation models with respect to performance optimization from both service user point of view as well as service provider point of view.

4.9 Report I – NxET Reasoning Machine

Paramai Supadulchai

Plug-and-play Technical Report, Department of Telematics, NTNU

ISSN 1500-3868.

Abstract

Native XML Equivalent Transformation (NxET) is an XML-based implementation of the ET paradigm (ET = Equivalent Transformation). NxET is designed for simplicity, modularity, extensibility, mobility and performance and can be used as a reasoning machine in various application domains. NxET has been used as supplemented reasoning functionality for the adaptable service systems based on TAPAS. This article focuses on NxET's computing paradigm, actor model, data model, and NxET implementation and usages. NxET's computing paradigm consists of a conceptual modeling language and a computation model. The actor model describes the use of NxET with actors in the TAPAS computing architecture. The data model describes XML-based data representations of the conceptual modeling language. The NxET implementation and usages illustrates NxET system components, processing mechanism and usages.

Contributions

C1{RM functionality}, **C2**{Generic reasoning model, RM-based role figures}, **C4**{EFSM and RM-related data entities, RM-based specification representation}

Summary

This technical report presents NxET Reasoning Machine that is the implementation of the generic reasoning model and RM-based role figures. The presented issues are as follows:

- I) NxET's computing paradigm
- II) The actor model
- III) The data model
- IV) NxET implementation and usages

This technical report does not provide the TAPAS concepts, TAPAS service functionality architecture as well as TAPAS computing architecture. The readers should refer to Section 2 and Section 3 before reading this report.

NxET's computing paradigm is denoted as XET computing paradigm and consists of a conceptual modeling language and a computation model. The conceptual modeling language is based on XML Declarative Description (XDD). However, XDD are just data representation. A computation model is needed. The XET computation model is based on Equivalent Transformation (ET) [Aka98], which solves a given problem by transforming it through repetitive application of (semantically) equivalent ET transformation rules. An XDD clause is modeled by an ET transformation clause.

The actor model presents the integration of RM functionality in the capability-based computing architecture. The actor role in the TAPAS computing architecture is defined as an Extended Finite State Machine (EFSM) extended with policies. The mechanism interpreting the manuscript is an EFSM interpreter extended with a reasoning machine. The data entities of EFSM and RM are presented.

The data model presents the XML-based representation of the RM data entities consisting of the policies, system constraints, reasoning conditions, facts and transformation clauses. These data entities are represented by XML variables, XML expression, XET clause and XET rule. XML variables are used to represent implicit information. XML expressions are ordinary XML elements with the XML variables. XET clauses are XML-based representation of the ET

transformation clauses. XET rules are XML-based representation of the ET transformation rule.

The NxET implementation and usages section presents NxET system components, NxET processing mechanism and NxET usages. The NxET system components are the components constituting the NxET reasoning machine. The NxET processing mechanism shows the implementation details of the NxET reasoning machine. The usages shows that NxET can be used as either a standard application, a procedural call service within a Java application and by using XET Rule Editor.

4.10 Relationship and overlap between the publications

The relationship between Paper A – Paper H as well as Report I is illustrated in Figure 19.

Paper A's main focus is the concept model and data representation, which is applied in the work in all other papers, and the ontology-based capability selection mechanism. Paper B main focus is the capability initialization and re-initialization. Paper C is an extension of Paper B and provides a generic policy system definition for the capability initialization. Paper D describes the RM-based role figures and demonstrates a potential use for the capability re-initialization. This paper has also contributed additional concept model and data representation extension and can be considered as a complement of Paper A. Paper E defines the formalism of the RM-based role figures given in Paper D.

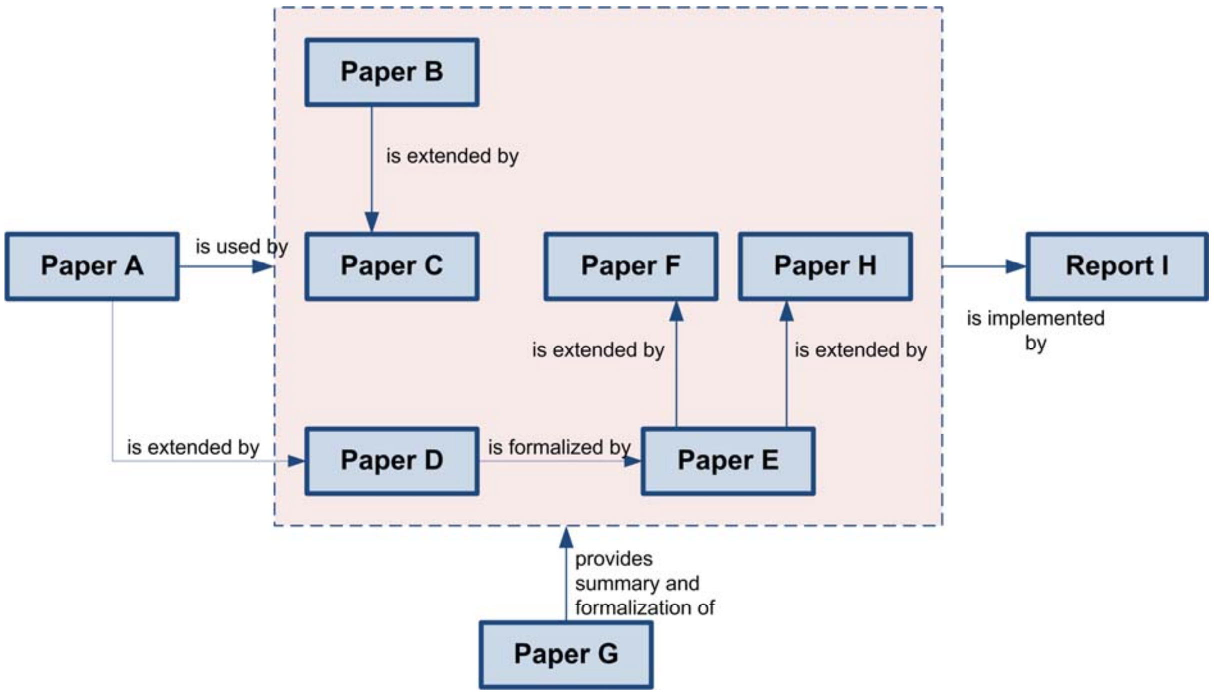


Figure 19 – The relation between the selected publications

Paper E, Paper F and Paper H give a contribution to the capability allocation adaptation. Paper F is an extension of Paper E and provides more experimental results and discussion related to the capability allocation adaptation. Paper H extends the capability allocation adaptation model in Paper E with the parameter production and SLA mapping.

Paper G presents a capability-based computing architecture and provides a summary and improvements of the concepts, models and mechanisms in Paper A – Paper F.

Report I is about the implementation of the reasoning machine and has relationships with all other papers.

5. Related works

This section presents related researches, technologies and platforms. The presentation is structured in two main parts: 1) related paradigms and 2) research papers. The paradigms presented are:

- Autonomic systems and autonomic communications
- Pervasive and ubiquitous computing
- Context-aware system

Even if these paradigms have overlapping objectives, concepts, solutions, features and mechanisms, the focus is considered to be different. Concerning the research papers, the discussion of these are based on the contributions C1-C4 defined in the previous sub-section.

5.1 Related paradigms

5.1.1 Autonomic Systems and Autonomic Communications

Autonomic systems and Autonomic communications are emerging technologies inspired by the autonomic computing paradigm introduced by IBM [Kep03, Str04]. Autonomic systems are characterized by self-configuration, self-monitoring, self-adaptation and self-healing properties. The ultimate goal of *Autonomic communications*, however, is networks and associated devices and services that will be able to work in a totally unsupervised manner; able to self-configure, self-monitor, self-adapt and self-heal [Dob06].

Autonomic elements are building blocks of an autonomic system [Fua06] where the elements themselves are self-contained and facilitated with the *self-x* properties. Feedback loops are usually visible either within an autonomic element itself [Fau06] or among them [Dob06]. Techniques such as cognitive reasoning [Nar03], [Nb04], [Ser06], policy management, [Nb04], intelligent agents [Zha04], biological algorithms [Sus05], context-awareness [Ser06] and semantic annotations [Ser06] are applied.

Autonomic communication and adaptable service systems share the same direction towards the handling of complexity introduced by the higher degree of flexibility required. On the architectural perspective, the autonomic communication can be seen as a subset of adaptable service systems. This is because the autonomic communication sets strong restrictions to the architecture solutions applied.

The focus of the work in this thesis can be related to self-configuration, self-optimization and self-healing with respect to capabilities. The self-protection, on the other hand, is not considered.

5.1.2 Pervasive and ubiquitous computing

Pervasive computing considers information processing, retrieval and utilization to be integrated into everyday life. The goal of *ubiquitous computing* concept is similar to the pervasive computing concept. However, it focuses more on a person and his or her activities with different kinds of computing devices. The use of computing devices shall be so integrated in the personal activities that they are almost invisible. *Adaptive pervasive computing* considers changes in the environment around a user and the ability of the service system to handle the changes. Techniques such as policy-based management [Har05], [Han06] and [Syu07], cognitive reasoning [Lew04] and semantic service composition [Lew04] may be applied.

Considering the adaptability properties defined, the pervasive and ubiquitous computing has its main focus on the core property **A1** – rearrangement flexibility. The pervasive and ubiquitous computing consider QoS and performance with respect to the users. A service system in this paradigm is of course required to have both failure robustness and resource load awareness and control. This is however not the focus problem to be solved within these paradigms. While Adaptable service systems consider QoS and performance from both service provider and service users' point of view, pervasive and ubiquitous computing, however, consider QoS and performance with respect to the users.

5.1.3 Context-aware Systems and Services

Context-aware systems are the system that can sense the surrounding environment and aggregate the information as "*context*". A context can consist of the information such as user, terminal, terminal capabilities, geographical position, nature of the position (shop, work, nature, etc.), events, failures and performance. A definition of context given by [Dey01] is “any information that can be used to characterize the situation of entities”. A context aware service system provides user services dependent on the context.

The mechanism used by context-aware systems can be based on reasoning machine [Dan07] and ontology [Pes07], which are similar to the sub-contributions in this PhD work. However, the context-aware systems focus more on providing services when the context is changed. Unlike adaptable service systems, the restructuring of the service system and service components to realize the core properties is not the main focus.

5.2 Related papers

The following discussion categorized and discussed with reference to the contributions **C1-C4** defined in the previous section.

5.2.1 Contributions C1 and C4

The capability-based computing architecture (C1) and the concept model and the data representation contributions (C4) are usually found in the same architectural framework. Therefore, these two contributions are considered in this section.

The following sub-contributions of **C1** and **C4** will be discussed.

- Capability (C1)
- Capability performance (C1)
- Service performance (C1)
- SLA (C1)
- RM-based functionality (C1)
- Concept ontology (C4)
- RM-based data representation (C4)
- SLA mapping (C1, C2, C3)

Five capability-based computing architectures [Jøa01], [Viv03], [Dur05], [Xia05], [Adl06] are considered. These architectures are intended to support QoS in networked service systems. A system is considered to have the SLA contribution if the system has the ability to classify users and allocate capabilities based on SLA. SLA mapping illustrates the ability to autonomously transform high-level SLA into lower level operational concepts such as capability requirements, capability performance requirements and service performance requirements.

Aagedal published a QoS support in development of distributed systems in his thesis [Jøa01] in 2001. The work appropriately defines QoS concepts that are necessary to model distributed multimedia applications in both conceptual level and implementation level. A Common Quality Modeling Language (CQML) is proposed as a lexical language for QoS ontology modeling. The thesis represents how CQML can be used and how the models, based on CQML, can be compiled by a computational support into QoS templates that will be used in multimedia application systems. The QoS specification, however, is made per user instead of classes. The work does not have the RM specification and execution integrated.

[Viv03] provides a QoS architecture for effective pricing model. The architecture focuses on three aspects: 1) QoS classification, 2) pricing scheme definition and 3) algorithm for performance optimization with respect to the service user and service provider point of views. The architecture does not have a fully featured capability-based computing architecture. Only the service performance is considered. The QoS specification is defined directly in the physical level. The RM concept as well as the concept model and data representation is not considered.

[Dur05] presents an End-User Specification of Quality of Service Applying the Model-Driven Approach. The architecture aims to provide flexibility. UML is used to model capability, capability performance, service performance and SLA in a conceptual level. The QoS concepts, once defined, can be translated into an equivalent XMI document, which is available in XML format. Based on the XMI documents, an XMI parser creates low-level QoS-related requirements for operational applications. The XMI parser does not use policies for the creation of the task. The architecture is therefore considered without the RM-based functionality.

[Xia05] presents a QoS-aware service composition and adaptation in autonomic communication. Similar to [Viv03], the architecture does not have a computing architecture and the QoS-related requirements are specified directly in the physical level. The ontology concept is not applied in the architecture. The only considered QoS concept is the service performance. The architecture has a domain graphs that, based upon the service performance measures, can determine the effective route between two domains. The paper's main focus is the service performance optimization between two domains, where a domain can be seen as a service provider and the other domain can be seen as a service user.

The work in [Adl06] presents a programming environment that has QoS control for grid applications. The architecture uses the notion autonomic element following the concept in Autonomic Computing. The architecture has an abstraction level where capability, capability performance and service performance can be defined. These concepts can be recognized by application managers in order to configure and optimize the system accordingly. The autonomic elements have RM-based functionality. The RM-based data representation, however, is not defined. The concept SLA is also not visible in the architecture.

Table 4 illustrates a comparison of the related work with respects to the capability-based computing architecture related work.

Table 3 – Capability-based computing architecture related work

Sub-contributions	[Jøa01]	[Viv03]	[Dur05]	[Xia05]	[Ald06]
Capability	Yes	-	Yes	-	Yes
Capability performance	Yes	-	Yes	-	Yes
Service performance	Yes	Yes	Yes	Yes	Yes
SLA	Yes[2]	Yes	-	Yes[3]	-
Concept ontology	Yes	-	Yes	-	Yes
RM-based functionality	-	-	-	-	Yes
RM-based data representation	-	-	-	-	-
SLA mapping	Yes	-	Yes	-	-

[2] SLA is made per user, not a group of users.

[3] SLA is made between two domains.

5.2.2 Contributions C2

The following sub-contributions of the contribution C2: Policy-based reasoning will be discussed.

- RM-based role figures
- Generic policy system definition
- Ontology-based capability selection mechanism
- Dynamic policies
- Parameter production

Five papers [Gar04], [Agra05], [Sha05], [Har06] and [Syu07] are considered. The work in all these papers has a generic reasoning model. Concerning the RM-based role figures, a work is considered having the contribution related to the RM-based role figures if the policy-based reasoning functionality can be distributed to system components; i.e. not a centralized architecture.

[Gar04] presents an architecture-based self-adaptation framework with reusable infrastructure. The aim of the architecture is to reduce difficulties when adding a new service component into the system. This is done through annotating common knowledge for service component, which can be aligned with the generic policy system definition. The capability ontology, however, is not applied. The framework neither has the dynamic policies feature nor the parameter production feature.

[Agra05] presents a middleware for policy management for networked systems and applications. The middleware provides an infrastructure for creation, storage, distribution and execution of policies. The architecture has three focuses: 1) a general policy system definition, 2) local administration of policies and 3) policy transformation. The local administration of policies feature allows system administrators to accept, reject or flag policies. Thus, the dynamic policies feature of this architecture partially requires human intervention. The policy transformation allows a new policy to be transformed based on the current situation.

[Sha05] presents a dynamic policy framework that automatically creates policies for differentiated communication systems. A hierarchical policy model is used to capture users and administrators' higher level goals and transform them into network level policies, which will be distributed to the service components. The framework focuses mainly on the policy composition. The generic policy system definition and the ontology model are not applied.

[Har06] provides a policy-based context-aware agent framework supporting users' mobility. Policies to deal with users' rearrangement flexibility and the capabilities of mobile devices can be composed and distributed. This work has a focus on the capabilities needed by the users. [Sha05] and [Har05] share similar policy composition frameworks.

[Syu07] defines an architecture for policy-based control of context aware pervasive services. The architecture has a policy mechanism to control context-aware behavior for pervasive service applications. The targeted service application is selecting policies to apply for mobile users in a pervasive environment based on context. The concepts included in the context are system performance, resources, rearrangement flexibility requirements and dependability requirements. A proprietary XML-based policy language and a reasoning engine are defined and implemented. The architecture has, in addition, ability for policy conflict resolution. The main difference of this architecture and this thesis is the focused service system applications. Similar to [Han06], [Syu07]'s application focuses on mobile users. This thesis, however, focuses on the adaptability properties of service systems.

Table 4 illustrates a comparison of the related work [Gar04], [Agr05], [Sha05], [Han06] and [Syu07] with respects to the policy-based reasoning sub-contributions.

Table 4 – Policy-based reasoning related work comparison

Sub-contributions	[Gar04]	[Agr05]	[Sha05]	[Han06]	[Syu07]
RM-based role figures	Yes	-	Yes	Yes	-
Generic policy system definition	Yes	Yes	-	-	-
Ontology-based capability selection mechanism	-	-	-	-	-
Dynamic policy	-	Yes	Yes	Yes	-
Parameter production	-	-	-	-	-

5.2.3 Contributions C3 and C2

This section has main focus on the contribution C3: Capability configuration management. Some of the sub-contributions of the contribution C2: Policy-based reasoning, are also included. This is done because the structure of the content of the research papers considered. The following sub-contributions of the contribution C3 and C2 are discussed.

- Capability initialization and re-initialization (C3)
- Capability allocation adaptation (C3)
- Capability allocation adaptation with dynamic policies (C3 and C2)
- Capability allocation adaptation with parameter production (C3 and C2)
- Ontology-based capability selection mechanism (C2)

Five most related papers [Xu00], [Ben01], [Sah04], [Cor06] and [Alm06] are considered. The service configurations in these architectures are all based on capability, capability performance and service performance.

[Xu00] presents a multimedia service configuration and reservation in heterogeneous environments. A capability initialization is used to compose a service based on chosen service components to clients. Service reservation has a role of capability allocation adaptation to optimally reserve resources for end-to-end communication with the end users. The architecture can compose and re-compose the service based on capability and service performance data. A capability allocation adaptation model is provided. However, the architecture does not support the use of ontology, the dynamic policies and the parameter production.

[Ben01] proposes a framework for conceptually defining a service model and a composition model for composing the service instance. The framework has a mechanism similar to the dynamic policies. The service model can be adapted to changes in the environment and the service instance can be re-deployed. The framework focuses on the service composition between two domains. Capability and capability performance requirements are encapsulated in negotiation objects, which both domains must agree on. The framework does not support the use of ontology models for the service composition.

The architecture in [Sah04] has automated policy-based capability composition functionality. A composite capability is composed by one or more based capability. This paper considers a composite capability as a service. By using policies and a reasoning technique, a new composite capability can be composed from base capabilities. The policies, however, cannot be adapted. The framework focuses mainly resource composition but not re-composition and adaptation. The concept model representing policies, capabilities and capability ontology is based on Common Information Model (CIM), which is also used to represent physical capabilities in this thesis.

[Cor06] considers a context-aware architecture that can dynamically compose services based on capability profiles of portable devices, user profiles and service profiles. The architecture model these profiles and composition policies dynamically using Web Ontology Language (OWL). The ontology models can be adapted to changes in the environment. When new mobile devices are introduced, the mobile ontology, for example, will be adapted. The framework focuses mainly service composition with respect to rearrangement flexibility. The service adaptation and optimization is out of scope of the architecture.

[Alm06] focuses on capability allocation adaptation issues with respect to capacity and performance in an autonomic service-oriented architecture. From the service provider point of view, services provided to service users are constituted by Web Services, which requires capabilities to execute. A capability planning and optimizing mechanism is proposed. The mechanism can optimize the capabilities required by each Web Service class based on cost functions of Web Service class as well as the usage of Web Service instances. The novelty of the architecture resides in the ability of the planning and optimizing mechanism to create both short-term and long-term plan for superior capability allocation adaptation.

Table 5 illustrates a comparison of the related work [Xu00], [Ben01], [Sah04], [Cor06] and [Alm06] with respect to the capability configuration management.

Table 5 – Capability configuration management related work comparison;

Sub-contributions	[Xu00]	[Ben01]	[Sah04]	[Cor06]	[Alm06]
Capability initialization and re-initialization	Yes	Yes	Yes*	Yes	-

Capability allocation adaptation	Yes	Yes	-	-	Yes
Ontology-based capability selection mechanism	-	-	Yes	Yes	-
Capability allocation adaptation with dynamic	-	Yes	-	Yes	-
Capability allocation adaptation with parameter production	-	-	-	-	-

6. Summary, conclusions and further work

6.1 Summary

The context of this thesis has been capabilities in adaptable service systems. The main research objectives were "to study and analyze concepts, models and mechanisms for the realization of the general and the core properties related to capabilities, and further to specify, construct, evaluate and validate a framework for capability configuration management." Based on these research objectives as well as the general and core properties defined in Section 1.1, and the scope defined in Section 2.2, six problem statements P1-P6 was defined in Section 2.2.

A solution framework denoted as a *reasoning-based capability configuration management framework for adaptable service systems* is intended to meet the objectives as well as the problem statements defined. The solution framework, which was presented in Section 3, consists of the following contributions:

C1: Capability-based computing architecture,

C2: Policy-based reasoning,

C3: Capability configuration management,

C4: Concept model and data representation and

C5: Scenarios – experimentation and simulation.

These contributions consist of sub-contributions. Each of the sub-contributions C1-C4 represents contributed concept, model or mechanism for the realization of the *adaptability properties* as defined in Section 1.1 The contribution C5 is intended to evaluate and validate the contributions C1-C4. The contributions C1-C5 was presented in Section 3.1.The relationship between the contributions and the publications presented in part II was presented in Section 4. *A discussion of how the various contributions implement the problem statements was presented in Section 3.2.*

Concerning contribution C5: Scenarios – experimentation and simulation, the following five scenarios S1-S5 were designed and used for evaluation and validation:

S1: Ontology-based capability selection for a Tele-school application

S2: Printing system capability initialization

S3: Capability re-initialization of RM-based role figures

S4: Policy-based capability allocation adaptation for streaming service,

S5: Policy-based parameter production for policy-based capability allocation adaptation.

6.2 Conclusions

The solution framework presented in this thesis consists of contributions C1-C5 that aim to meet the problem statements:

P1: How to include capabilities as well as additional needed concepts in the architectural framework?

P2: How to include capability management in an architectural framework?

P3: Which concepts, features and mechanism are appropriate for autonomic capability management?

P4: What is the ontology to be used? How to model and represent the concepts of the ontology?

P5: How to model capability management functionality when considering modularity, reusability, expressive power and flexibility?

P6: How to evaluate and validate the proposed framework?

The evaluation and validation of the framework (Contribution C5) by experiments and demonstrations concludes that the framework meets the problem statements. The contributions have been included in the TAPAS computing and service architecture.

The solution framework is however a generic framework consisting of concepts, models and mechanisms to be re-used. The framework does not give any method to specify policy based adaptable functionality. The lessons learnt from the design, specification and execution of the policies for the policy generation in scenario S4 and parameter production in scenario S5 is that the design of policies needs accurate knowledge of the application to be managed. The policy development process must follow a traditional iterative software life-cycle process. The goodness of the specific policy system must be validated, evaluated and optimized in each case. Using policies introduced flexibility with respect to adding new features fast and easily. The goodness of policy system must be evaluated and validated based on traditional methods.

6.3 Further work

Adaptable service systems represent a vision – this even if the scope of the definition is executing adaptable systems. Considering the life cycle of a service it will include both a human part and an autonomous part. The human part represents design, specification and maintenance, while the autonomous part mainly is adaptation during execution. The human part defines and improves the behavior of service systems. The autonomous part uses the defined behavior to make the service systems adaptable and generates feedbacks for human to improve the behavior applied. The adaptability of service systems relies on the knowledge captured from the situations that have happened in the past. Although the system has ability to learn, capturing new knowledge without human intervention is not trivial. For example, policies controlling the behavior of the service systems in the presented scenarios have been defined by human. While the evaluation of the consequences of the policies is possible, more research and development efforts are required to define new policies autonomously.

Concerning the evaluation and validation of the solution framework presented the models and mechanisms have not been formally validated by formal methods, and there has not been any comprehensive analysis of the policies applied in the scenarios S4 and S5. Only demonstration and simulation have been applied, and these demonstrations and simulations are handling example cases. The outcome is then rather an indication of the potential and applicability than a proof of the applicability. It can neither be ensured that the models and mechanisms will always lead the service system to desired states nor that the models and mechanisms are optimal in all situations. A more accurate evaluation and validation should be conducted.

The solution framework assumes the availability of the other functionality in the TAPAS service architecture. It is assumed that the capability and service performance administration, mobility management as well as service management are implemented and can be applied with the capability configuration management. More efforts are needed to design and implement the capability and service performance administration as well as the service management functionality. The integration of all functionality must also be evaluated and validated.

Concerning the aspects of the capability configuration management framework which was outside the scope of this thesis, these aspects should be included. The capability management framework should both be based on a distributed architecture and protocol and it should fulfill the failure robustness core properties.

Finally, limitations related to the used programming language, tools, programming and algorithms must be taken into accounts. For examples, the current reasoning machine implementation depends on the full version of JRE. Thus, the solution framework is limited to nodes with sufficient capabilities to execute the full version of JRE. Some modifications are needed to execute the RM-based functionality on the mobile version of Java (J2ME).

Bibiography

[Aag01a] F.A. Aagesen, B.E. Helvik, U. Johansen and H. Meling, *Plug and Play for Telecommunication Functionality* -- *Architecture and Demonstration Issues*, The International Conference on Information Technology for the New Millennium (IConIT), Thammasat University, Bangkok, Thailand, May 2001.

[Aag01b] F. A. Aagesen, *The PaP project - Research method*, *Plug-and-Play Technical Report*, Department of Telematics, NTNU, 2001-02-01, ISSN 1500-3868.

[Aag02a] F.A. Aagesen, B.E. Helvik, and C. Anutariya, *Towards Dynamic Composition of Adaptive Services*, in Proceedings of EuroWeb 2002 Conference -- The Web and the Grid: From E-science to E-business, Oxford, UK, December 2002.

[Aag02b] 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.

[Aag03] F. A. Aagesen, B. E. Helvik, C. Anutariya and M. M. Shiaa, *On Adaptable Networking*, in Proceedings of the First International Conference on Information and Communication Technologies, ICT'2003, Assumption University Thailand, April 2003.

[Aag05] F.A. Aagesen, P. Supadulchai, C. Anutariya and M.M. Shiaa, *Configuration Management for Adaptable Service Systems*, in Proceedings of IFIP International Conference on Metropolitan Area Network, Architecture, Protocols, Control and Management (MAN 2005), Ho Chi Minh City, Vietnam, 2005.

[Aag07] F.A. Aagesen and P. Supadulchai, *A Capability-based Service Framework for Adaptable Service Systems*, submitted to The 2nd International Conference on Advances in Information Technology (IAIT2007), Bangkok, Thailand, 2007.

[Aag99a] F. A. Aagesen, B. E. Helvik, H. Meling and U. Johansen. *Plug and Play for Telecommunications -- Architecture and Demonstration Issues*, Norsk Informatikkonferanse, Trondheim, November 1999, Tapir, ISBN 82-519-1556-2.

[Aag99b] F. A. Aagesen, B. E. Helvik, V. Wuwongse, H. Meling, R. Bræk and U. Johansen, *Towards a Plug and Play Architecture for Telecommunications*, in Proceedings of the IFIP TC6 WG6.7 Fifth International Conference on Intelligence in Networks, Asian Institute of Technology, Thailand, November 1999.

[Adl06] M. Aldinucci, M. Danelutto, M. Vanneschi, *Autonomic QoS in ASSIST Grid-aware components*, in Proceedings of 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Computing (PDP'06), Montbéliard-Sochaux, France, February 15-17, 2006.

[Aka98] K. Akama, T. Shimitsu, and E. Miyamoto, *Solving Problems by Equivalent Transformation of Declarative Programs*, Journal of the Japanese Society of Artificial Intelligence, vol. 13, pp. 944-952, 1998.

[Alm06] J. Almeida, V. Almeida, D. Ardagna, C. Francalanci, M. Trubian, *Resource Management in the Autonomic Service-Oriented Architecture*, in Proceedings of ICAC '06. IEEE International Conference on Autonomic Computing, Dublin, Ireland, 13-16 June 2006.

[Alo03] G. Alonso, F. Casati, H. Kuno, V. Machiraju, Web Serices, Springer, ISBN 978-3540440086, 2003.

[Agr05] D. Agrawal, S. Calo, J. Giles, K. Lee and D. Verma, *Policy Management for Networked Systems and Applications*, in Proceedings of the 2005 9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005), pp. 455 – 468, Nice, France, 15-19 May 2005.

[Ben01] I. Ben-Shaul, O. Holder, B. Lavva, Dynamic adaptation and deployment of distributed components in Hadas, IEEE Transactions on Software Engineering, Vol. 27, No. 9, Pp. 769 – 787, Sept. 2001.

[Bad06] B. Tebbani and I. Aib, *GXLA a Language for the Specification of Service Level Agreements*, in Proceedings of the First International IFIP TC6 Conference on Autonomic Networking (AN 2006), Paris, France, September 27-29, 2006.

[Bri91] E. Brinksma, What is the method of formal methods? IFIP Forte 91, Sidney, November 1991.

[Cha06] H. Chan and T. Kwok, *A Policy-based Management System with Automatic Policy Selection and Creation Capabilities by using a Singular Value Decomposition Technique*, in Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06), The University of Western Ontario, London, Ontario, CANADA, 2007.

[Cho05] P. Cholda, A. Jajszczyk, B.E. Helvik, A. Mykkeltveit, *Service Differentiation Based on Recovery Methods*, in Proceedings of the 2nd EuroNGI Workshop on Traffic engineering, protection and restoration for NGI, Rome, 21–22 April 2005.

[Cor06] A. Corradi; R. Montanari, A. Toninelli A, Dynamic configuration of semantic-based service provisioning to portable devices, in Proceeings of International Symposium on Applications and the Internet 2006 (SAINT 2006), Phoenix, Arizona, USA, 23-27 Jan. 2006.

[Dan07] L. Daniele, P. D. Costa and L. F. Pires, *Towards a Rule-Based Approach for Context-Aware Applications*, in Proceedings of the 13th EUNICE Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services, University of Twente, the Netherlands,

[Day01] A.K. Dey, *Understanding and Using Context*, Personal Ubiquitous Computing, Vol. 5, No. 1, pp. 4-7, February 2001.

[Dia05] Y. Diao, J. L. Hellerstein, S. Parekh, R. Griffith, G. E. Kaiser and D. Phung, *A Control Theory Foundation for Self-Managing Computing Systems*. IEEE Journal on Selected Areas in Communications, Vol. 23, No. 12, December 2005.

[Dob06] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt and F. Zambonelli, *A survey of autonomic communications*, ACM Transactions on Autonomous and Adaptive Systems (TAAS), Vol. 1, No. 2, December 2006.

[Dit06] M. Ditze and T. Bresser, *Resource adaptation for audio-visual devices in the UPnP QoS architecture*, in The 20th International Conference on Advanced Information Net-working and Applications (AINA 2006), 2006.

[Dmt07a] DMTF, Common Information Model (CIM) Standards, available on-line at http://www.dmtf.org/standards/cim/, November 2007.

[Dmt07b] DMTF, xmlCIM, *CIM Tutorial*, available on-line at http://www.wbemsolutions.com/tutorials/CIM/wbem-xmlcim.html, November 2007.

[Dur05] D. Durand and Christophe Logé, *End-User Specification of Quality of Service Applying the Model-Driven Approach*, in Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services (ICAS/ICNS 2005), Papeete, Tahiti, 23-28 October 2005.

[Fua06] M.M. Fuad and M.J. Oudshoorn, *System Architecture of an Autonomic Element*, in Proceedings of the Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASe'07), Baltimore, MD, USA, March 2007.

[Gar04] D. Garlan, S. Cheng, A. Huang, B. Schmerl and P. Steenkiste, *Rainbow: Archiecture-Based Self-Adaptation with Reusable Infrastructure*, Computer, pp. 46-54, October 2004.

[Han06] S. Han, S. Zhang, Y. Zhang, *Self-Adaptive Pervasive Computing Application Based on Code Mobility*, in Proceedings of the 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing, Indianapolis, Indiana, USA, September 29-October 1, 2006.

[Har05] H. Harroud and Ahmed Karmouch, *A Policy Based Context-aware Agent Framework to Support Users Mobility*, in Proceedings of the Ninth IFIP/IEEE International Symposium on Integrated Network Management (IM 2005), Nice, France, 15-19 May 2005.

[Ino99] Y. Inoue, et al., *The TINA Book: A Co-operative Solution for a Competitive World*: Prentice Hall, 1999.

[ITU02] ITU-T, *ITU-T Recommendation E.860, Framework of Service Level Agreement*, 2002.

[ITU92] ITU-T, *Principles of intelligence network architecture*, October 1992.

[Jia03] S. Jiang and F. A. Aagesen, *XML-based Dynamic Service Behaviour Representation*, Proceedings of *NIK'2003*, Oslo, Norway, 2003.

[Jia06] S. Jiang and F. A. Aagesen, *An Approach to Integrated Semantic Service Discovery*, Proceedings of *Autonomic Networking 2006*, Paris, 2006, pp. 159-171.

[Jia07] S. Jiang and F. A. Aagesen, *Efficient Service Discovery System Based on Semantic Overlay Networks*, Proceedings of *2007 SIWN International* Conference *on Complex Open Distributed Systems (CODS'2007)* Chengdu, China, 2007.

[Jøa01] Jan Øyvind Aagedal, *Quality of Service Support in Development of Distributed Systems*, PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, 2001.

[Kep03] J. O. Kephart and D. M. Chess, *The Vision of Autonomic Computing*, Computer, Vol. 36, No. 1, pp. 41-50, Jan 2003.

[Lew04] D. Lewis, O. Conlan, D. O'Sullivan and R. Wade, *Managing adaptive pervasive computing using knowledge-based service integration and rule-based behavior*, in Proceedings of IEEE/IFIP Network Operations and Management Symposium 2004 (NOMS 2004), Vol. 1, Pp. 901 – 902, Seoul, South Korea, April 2004.

[Liu06] H. Liu and M. Parashar, *Accord: A Programming Framework for Autonomic Applications*, IEEE Transactions on Systems, Man and Cybernatics – Part C: Applications and Reviews, Vol. 36, No. 3, May 2006.

[Lüh04] E. Lühr, *Mobility support for wireless devices - within the TAPAS platform*, MSc thesis, Department of Telematics, NTNU, 2004.

[Mel06] H. Meling, *Adaptive Middleware Support and Autonomous Fault Treatment: Architectural Design, Prototyping and Experimental Evaluation*, Doctoral Thesis, Department of Telematics, Norwegian University of Science and Technology (NTNU). Trondheim, 2006.

[Nar03] S. Narain, T. Cheng, B. Coan, V. Kaul, K. Parmeswaran, W. Stephens, *Building autonomic systems via configuration*, in Proceedings of Autonomic Computing Workshop 2003, Pp. 77 – 84, 25 June 2003.

[Nb04] N. Badr, A. Taleb-Bendiab, D. Reilly, *Policy-based autonomic control service Policies for Distributed Systems and Networks*, 2004.in Proceedings of the Fifth IEEE International Workshop on POLICY 2004, IBM Thomas J Watson Research Center, New York, pp. 99 – 102, 7-9 June 2004.

[Pes07] R. Pessoa, C. Calvi, J. P. Filho, C. Farias, R. Neisse, *Semantic Context Reasoning Using Ontology Based Models*, in Proceedings of the 13th EUNICE Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services, University of Twente, the Netherlands, July 2007.

[Sah04] A. Sahai, S. Singhal, R. Joshi and V. Machiraju, *Automated Policy-Based Resource Construction in Utility*, Computing Environments. IEEE/IFIP Network Operations and Management Symposium (NOMS'2004), Seoul, South Korea, 2004.

[Sam05] N. Samaan and A. Karmouch, *An Automated Policy-Based Management Framework for Differentiated Communication Systems*, IEEE Journal on Selected Areas in Communications, Vol. 23, No. 12, December 2005.

[Ser06] J. Martín Serrano, Joan Serrat, John Strassner and Ray Carroll, *Policy-based Management and Context Modelling Contributions for Supporting Services in Autonomic Systems*, in Proceedings of the First International IFIP TC6 Conference on Autonomic Networking (AN 2006), Paris, France, September 27-29, 2006.

[Sha05] N. Samaan and A. Karmouch, *An Automated Policy-based Management Framework for Differentiated Communication Systems*, IEEE Journal on Selected Areas in Communications, Vol. 23, pp. 2236-2247, 2005.

[Shi05] M. M. Shiaa, *Mobility Management in Adaptable Service Systems*, Doctoral Thesis, Department of Telematics, Norwegian University of Science and Technology (NTNU). Trondheim, 2005.

[Str04] J. Strassner, 2004 *IEEE/IFIP Network Operations and Management Symposium (NOMS 2004) Tutorial 7* - Autonomic Networking - Theory and Practice," 2004.

[Sup04] P. Supadulchai and F.A. Aagesen, *An Approach to Capability and Status Modeling*, in Proceedings of Norwegian Informatics Conference (NIK 2004), Stavanger, Norway, 2004.

[Sup05a] P. Supadulchai and F.A. Aagesen, *A Framework for Dynamic Service Composition*, in Proceedings of 1st International IEEE Workshop on Autonomic Communication and Computing (ACC 2005), Taormina, Italy, 2005.

[Sup05b] P. Supadulchai and F.A. Aagesen, *Autonomic Service Configuration by a Combined State Machine and Reasoning Engine-based Actor*, in Proceedings of the 2005 IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM 2005), Delta Centre-Ville Hotel, Montréal, Canada, 2005.

[Sup07a] P. Supadulchai and F.A. Aagesen, *Policy-based Adaptable Service Systems Architecture*, in Proceedings of the IEEE 21st International Conference on Advanced Information Networking and Applications (AINA-07), Niagara Falls, Canada, 2007.

[Sup07b] P. Supadulchai and F.A. Aagesen, *Towards Policy-Supported Adaptable Service Systems*, in Proceedings of the 13th Eunice Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services, University of Twente, the Netherlands, 2007.

[Sup07c] P. Supadulchai and F.A. Aagesen, *Autonomous Production of Parameters of an Autonomous Capability Allocation Adaptation Model*, Prepared for a submission, 2007.

[Sup07d] P. Supadulchai, *NxET Reasoning Engine*, Plug-and-play Technical Report, Department of Telematics, NTNU, ISSN 1500-3868.

[Sus05] J. Suzuki, T. Suda, *A middleware platform for a biologically inspired network architecture supporting autonomous and adaptive applications*, IEEE Journal on Selected Areas in Communications, Vol. 23, No. 2, pp. 249-260, Feb 2005.

[Syu07] E. Syukur and S. W. Loke, *Policy-based Control of Context Aware Pervasive Services*, Journal of Ubiquitous Computing Intelligence, Vol. 1, No. 1, pp. 110-131, 2007.

[Ten97] D. L. Tennenhouse, et al., *A Survey of Active Network Research, IEEE Communications Magazine,* vol. 35, 1997.

[Try05] T. Trygar and G. Bain, *A framework for service level agreement management*, Proceedings of *IEEE Military Communications Conference, 2005. MILCOM 2005.*, 2005.

[Vil04] J. J. Vila-Armengol, Implementation of Web services architecture in TAPAS, Msc Thesis, Department of Telematics, NTNU, June 2004.

[Viv03] D. A. Vivanco, R. Q. Kemp, and A. P. Jayasumana, *Effectiveness of Internet Pricing Models over a QoS Architecture*, in Proceedings. 28th Annual IEEE International Conference on Local Computer Networks (LCN '03), pp. 301-302, 2003.

[W3c04a] W3C, Resource Definition Framework (RDF) Primer, available on-line at http://www.w3.org/RDF/, February 2004.

[W3c04b] W3C, RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation, http://www.w3.org/TR/rdf-schema/, February 2004.

[W3c04c] W3C, OWL Web Ontology Language, W3C Recommendation, http://www.w3.org/TR/owl-features/, February 2004.

[Woo02] M. Wooldridge, *An Introduction to MultiAgent Systems*: John Wiley & Sons, Ltd., 2002.

[Xia05] J. Xiao and R. Boutaba, *QoS-Aware Service Composition and Adaptation in Autonomic Communication*, IEEE Journal on Selected Areas in Communications, Vol. 23, No. 12, pp. 2344-2360, December 2005.

[Xu00] D. Xu, D. Wichadakul; K. Nahrstedt, *Multimedia Service Configuration and Reservation in Heterogeneous Environments*, in Proceedings of the 20th International Conference on Distributed Computing Systems, Taipei, Taiwan, Pp. 512 – 519, 10-13 April 2000.

[Zha04] Y. Zhang, J. Sun and H. Ma, Self-management model based on multiagent and worm techniques, in Proceedings of Canadian Conference on Electrical and Computer Engineering, Sheraton Fallsview, Niagara Falls, Ontario, Canada, 2004

Part II – Selected publications

Paper A

An Approach to Capability and Status Modeling

Paramai Supadulchai and Finn Arve Aagesen

In Proceedings of *Norwegian Informatics Conference 2003 (NIK 2004)*

Stavanger, Norway, 2004

An Approach to Capability and Status Modeling

Paramai Supadulchai and Finn Arve Aagesen

Department of Telematics, Norwegian University of Science and Technology (NTNU)
N7491, Trondheim, Norway
{Paramai.Supadulchai, Finn.Arve.Aagesen}@item.ntnu.no

Abstract

A recent trend in network systems is the advanced technology to *dynamically* handle changes in the system. An important basis relies on the integration of *capability and status* representation and their semantic descriptions, which is currently not expressive enough. In this paper, we propose a *Unified Capability and Status Representation Framework (UniCS)* for handling several aspects related to capability and status. A scenario is modeled by UniCS to show how a network system exhibits adaptable behavior.

1 Introduction

Telematics Architecture for Play-based Adaptable System (TAPAS) uses a *Theater Metaphor* to model a network system in the same way a play is played in the theater. Anyone, who has acting skills, can become an *Actor* playing a *Role* in a *Play*. An Actor playing a Role is described as a *Role Figure*. The Role of an Actor is defined in a *Manuscript* containing the behavior of the Role Figure. An Actor is likely to become any Role Figure. However, factors such as sex, age, appearance or acting skill are the obvious barriers that allow some Role Figure to some Actors. These factors are the *Capabilities* of an Actor. It is the job of the *Director* of a Play to determine a fitting Actor to play a Role. In this task, the Director requests help from a *Service Management System*.

In a network system, *Nodes* are typically network processing units such as mobile phone, stationary computer, laptop, printer and router that possess particular *Capabilities*. At a specific time point, a *Status* is a system state with respect to the number of active entities, traffic situation and Quality of Service (QoS) etc. This applies to overall system as well as Nodes. The Capability and Status concepts will be discussed in Section 3.

Nodes have generic *Actors*, which normally inherit Capabilities and Status from the Nodes. The ability of an Actor to play a Role, which is modeled as an *Extended Finite State Machine* in a Manuscript [5], depends on the *Required Capability and Status* of the Role and the *Offered Capability and Status* in a Node where an Actor is going to play [1]. Employing an appropriate *Capability and Status Representation Framework* will help the Director and the Service Management System select the best Actor for a certain Role Figure. However, current Capability and Status representations lack well integration of the *Syntactic Representation* and the *Semantic Description*. This will lead to the difficulty when the Director does not understand clearly the meaning of a Capability or a Status. Though two Actors with slightly different Capabilities can play a specific Role interchangeably, the Director unfortunately does not have such knowledge and thus cannot assign the Role to a working Actor in case that the other one fails.

This paper presents an approach to capture and represent such knowledge in a formal way. Capability and Status are integrated with *Semantic Description* and *Configuration Rules* to enhance the reasoning process. Section 2 gives basic definitions of the *TAPAS Architecture*. An overview of Capability and Status is provided in Section 3. Next, Section 4 describes the definition of the *Play View Capability and Status*. The *Unified Capability and Status Representation Framework (UniCS)* is proposed in Section 5. In Section 6, a scenario demonstrates how to use UniCS to model a Play containing dynamic behavior of an adaptable network system.

2. TAPAS Architecture

The Telematics Architecture for Play-based Adaptable System (TAPAS) intends to be an architecture for adaptable network systems that gives rearrangement flexibility, failure *robustness*, QoS awareness and *resource control properties* [3]. In analogy with the TINA architecture principles, the architecture is separated in a *Service Architecture* and a *Computing Architecture*. The Service Architecture is an architecture showing the structure of *Services* and *Service Components*. The Service Architecture consists of *Primary Service Providing Functionalities* and additional *Service Systems*. These Service Systems are:

- *Service Management System*: definition of new Services, deployment and invocation of Services and Service Components
- *Capability Management System*: register, de-register, update, transform, provide access to Capabilities and manage Capability ontology.
- *Status Monitoring System*: provision of a view of the offered Status.
- *Configuration Management System*: optimization of Service Systems initial
- configuration and re-configuration with respect to Capabilities and QoS.
- *Mobility Management System*: The handling of various Mobility types.

The Computing Architecture is a generic architecture for the modeling of any Service Component. The Computing Architecture has three layers: *Service View*, *Play View* and *Network View*. The Service View works seamlessly with the Service Architecture to provide the modeling of an adaptable service. The Play View is the TAPAS specific concepts given in the introduction. The Network View concepts are the basis for implementing the Play View concepts, which again are the basic for implementing the Service View Concepts. In the other way around, the Service View concepts are mapped into the Play View concepts, which again are mapped into the Network View concepts. The Play View concepts are seemingly rearrangement flexibility oriented. The *Capability and Status* concepts, however, give a basis for the further design of the failure robustness, the survivability, the QoS awareness and the resource control properties.

In the Network View, Nodes are installed with the *Core Platform*, which has the execution support for the Play View concepts. Nodes publish their Capabilities and Status through a Capability Management System. However, the Capability Management System is beyond the scope of this paper. This paper covers mainly the representation of Capability and Status, which are managed by the Capability Management System and used by any Service System.

3. Capability and Status

We will provide an introduction to Capability and Status by showing the learning facilities of a school illustrated in Figure 1. A classroom is employed with TAPAS to enhance the adaptability of the classroom facilities. As illustrated in Figure 1, Node A is the smartphone of a student with a Bluetooth Capability. Laptop E has a Bluetooth Capability and a Wireless 802.11b Capability. These Nodes can connect to the network through Computer B using the Bluetooth Capability. In addition, the Laptop E can connect to the network via Wireless Access Point C.

Computer B has an Ethernet and a Bluetooth Capability. Computer B is installed with a TAPAS Director, a Service Management System and a Capability Management System. Computer D controls the printing jobs of Printer F and Plotter G. All Nodes are installed with the Core Platform and publish their Capabilities and Status through the Capability Management System. Students are assumed to use their own Nodes that have theWireless Capabilities such as Smartphone A and Laptop E. With these Capabilities, the students can remotely watch lectures live outside the classroom.

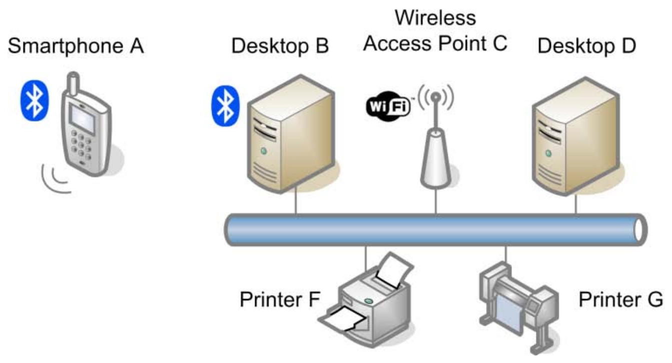


Figure 1: Teleschool environment in TAPAS

3.1 Capability

The general definition of Capability in TAPAS is an inherent property of a Node or a User, which defines the ability to do something. Capabilities can be classified as *Resource, Function and Data*:

- Resources: physical processing or storage components or transmission channels with finite capacity
- Functions: pure software or combined software/hardware functions, which perform particular tasks

- Data: just data, which interpretation, validity and life span depend on the context of the usage.

Table 1 shows an example of the Capability list of Computer B. Obviously, physical hardware components are categorized as Resources, which can be independent Nodes or dependent *Hardware Components*. The Hardware Components, such as CPU and memory, are devices that cannot offer functionalities independently and must be hosted by a Node. Functions are softwares that either work internally or provide external interfaces to other Nodes. The examples of the internal function and the external function are Operating Systems and Web Services respectively. Data are just logical information that is utilized or processed by functions before other Nodes can use. The Data shown in the Table 1 are a username and a password that are accessible from a specific user.

Capability	*Primitives*	*Variety*	*Arrangement*
CPU = Pentium(R) 2GHz	resource	replaceable	shared
Memory = 1 GB	resource	absolute	shared
Disk Storage = 100 GB	resource	absolute	shared
Network Card = 100 Mbit/s	resource	replaceable	shared
Bluetooth = USB 2.0	resource	absolute	shared
Capabilities = Speaker	resource	replaceable	exclusive
Operating System = Windows XP	function	absolute	exclusive
Web Server = Apache 2.0.46	function	replaceable	shared
Username/Password = john/*****	data	absolute	exclusive

Table 1: The list of capabilities of the device *Computer B* in the three primitive dimensions

A Capability can also be classified in a *Variety Dimension* whether it is *replaceable* or not. If a Capability is replaceable, it can be adjusted or replaced by another Capability with the similar properties. From Table 1, the CPU can be stepped down to save energy when battery is low. In addition, the Web Server can also be equally replaced with a new version when there is a functionality that has been fixed from security threats. In an *Arrangement Dimension*, a Capability can be shared or exclusive depending on *the visibility and the availability* to other Nodes. For example, the Nodes' CPU information must be visible. This allows the fastest Node to be selected. On the other hand, a password must be hidden. In addition, the Web Server provides a service exclusively to the local computers within the domain when it works in the exclusive mode. Otherwise, it provides the service to all computers in the shared mode.

3.2 Status

In addition to the Status definition given in the Introduction, Status and Capability are related in the way that a Capability always has Status but not the other way around. The Status of a Capability is considered the Capability state. A Status can be derived; for example, we can define that a computer is working off-line when either its network adapter is not functioning or the network is not working. This definition is denoted as a Configuration Rule saying that the off-line state of a computer is derived from the states of network and network adapter Capabilities. Another important aspect about Status is the value range, which is basically of two types: *symbolic or numeric*. The challenge is to provide unambiguous semantic to the value range: for instance, the Connectivity Status of a Node can be defined in only 2 distinct symbolic values: on-line and off-line. The semantic of the off-line state explains that a Node becomes offline when it has lost the connectivity for at least 5 minutes. *Quantifiable Status*, i.e. numerical status, can be defined from three definitions: the semantic of the upper/lower bound, the unit/precision and the impact when the value is changed. An example of the upper/lower bound semantic can be "the more harddisk space a Node has, the more possibility to install a new software". An example of the unit/precision semantic can be "a 100 GB disk is larger than a 100 MB one". An example of the value impact semantic can be "a program uninstallations are needed when harddisk space is used more than 80%".

3.3 The Ontological Capability and Status Framework

Capability and Status give extra knowledge to improve the reasoning power of the Director and the Service Management System. However, considering just the solid information about Capability and Status may limit the power of the reasoning mechanism. For example, if a printing system chooses a target printer based on printing-related capabilities, the system-may not be able to understand how plotter and printer are different. As a consequence, it would redirect a billing report to a costly plotter. Thus, it is not sufficient to define that two printers can be used interchangeably when they supply the same printing-related capabilities. Some constraints and taxonomy hierarchy are needed; for examples: "The cost-per-page difference of both printers must not exceed 10%." or "The plotter is categorized as high-resolution printer and the high-resolution printer is available exclusively to a large document.". These requirements constitute *Configuration Rules*, which can be used to make a common understanding of a specific Capability and Status. Configuration Rules also provide the relationship between well-defined Capabilities and Status. In this way, an *Ontological Capability and Status Framework* can be created.

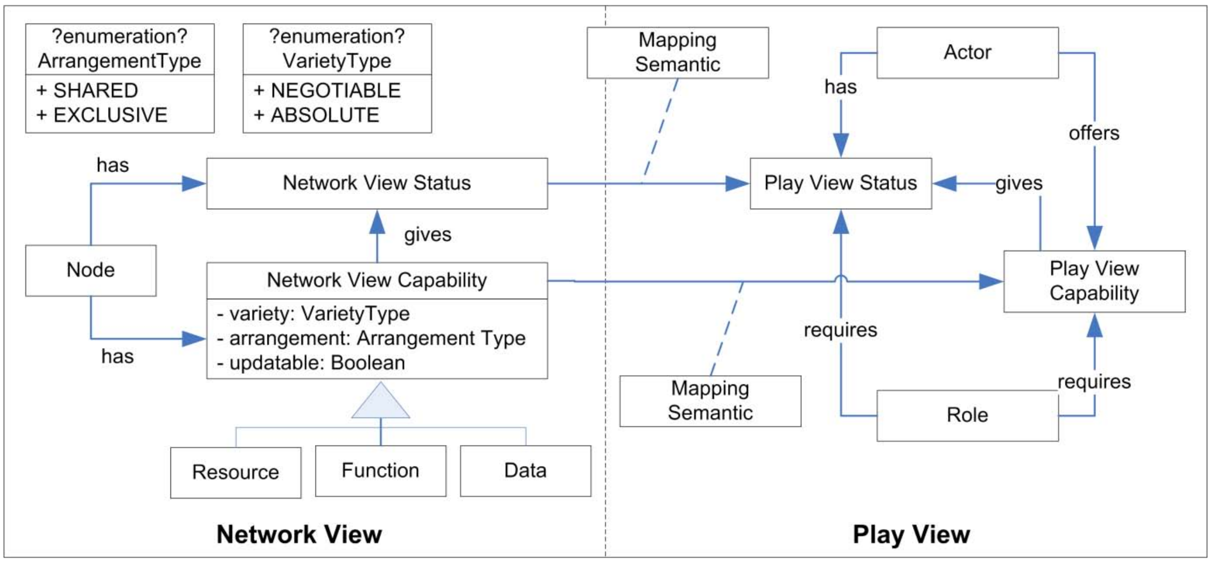


Figure 2: General Capability Model in TAPAS

4. The Play View Capability and Status

To properly perform the reasoning mechanism in adaptable networks, the proper understanding of Capability and Status is needed. A system designer can compose a *Play* to fulfill this requirement by combining several Configuration Rules that consist of *Capabilities*, *Status*, *Axioms* and *Constraints*. However, the system designer would have to design the Play that covers all possible Capabilities and Status of any Node. Alternatively, we propose the use of an abstraction layer of Capability and Status for the Play composition. Capability and Status are considered in the Network View and the Play View as illustrated in Figure 2. What up to now that has been considered as Capabilities and Status of a Node are denoted as Network View Capabilities and Status, abbreviated as NV-Capabilities and -Status respectively.

A Play View Capability, abbreviated as PV-Capability, is the function that an NVCapability offers in the Play View. It is an *abstraction* of an NV-Capability that

1. provides a unified representation with well-defined semantic to describe a Capability in the Play View,
2. maps two or more NV-Capabilities that potentially give the same functionality,
3. simplifies the *Play* creation process from the direct use of NV-Capabilities,
4. has specific meaning in a Domain.

Let's consider the Teleschool environment given in the Figure 1. Here we encounter two mobile Nodes with dissimilar Wireless NV-Capabilities. The *Smartphone A* uses Bluetooth to connect to other Nodes, while the *Laptop E* connects to the network with its 802.11b Wi-Fi. If the Director wants to execute a Play to detect all Nodes with Bluetooth or 802.11b NV-Capability, the Director will have to find out these Capabilities iteratively from all Nodes. If a student owns a laptop with a newer version of Wi-Fi 802.11g, the new NV-Capability *Wi-Fi 802.11g* must be included in the Play. Otherwise, the Play is not applicable to all possible Nodes.

Wouldn't it be nice if we can define shortly a Configuration Rule in the play "all Actors possessing wireless PV-Capability must turn off their Sound PV-Capability before entering a Domain that is offering classroom PV-Capability."? With the assumption that all NV-Capabilities can be mapped to PV-Capabilities, the play can be designed in a simpler manner from the PV-Capability requirement instead of using the NV-Capability directly. PV-Capabilities can be categorized into an ontology hierarchy; for example: wireless PV-Capability can be further separated to the long-range, mid-range and short-range transmission functions. The PV- and NV-concept are also applicable to Status the same way it is used for Capability. However, examples will not be given.

5. Unified Capability and Status Representation Framework (UniCS)

From earlier examples, we have given the examples of the Configuration Rule, which are used to formulate a Play. However, these rules are not meant to be performed by us. They shall be enforced by the network system. Therefore, a formal way to capture and make use of the Configuration Rules is required. In this section, a proposal for the formal representation of the NV-Capability and -Status and the PV-Capability and -Status is given in a unified framework denoted as the Unified Capability and Status Representation (UniCS).

5.1 Network View Capability and Status Representation

The Network View Capability and Status, abbreviated as NV-Capability and -Status, are used to describe Capability and Status extracted from Nodes. Their representation should provide a well-defined structure with precise semantic understandable by the network systems. Though any representation can be used with the NV-Capability and NV-Status because they will be eventually transformed into the PV-Capability and -Status, using a single standard accepted by all Domains is recommended. With a single standard representation, the system does not have to worry about how to read and understand the semantic of the representation since it is what everybody has agreed on. There are and will be a lot of ready-to-use tools and supports for the standard representation. In addition, only one set of Configuration Rules is required to map standardized NV-Capabilities and -Status into PV-Capabilities and -Status. Using a single representation therefore simplifies the integration of the instances of Capabilities and Status from many Domains.

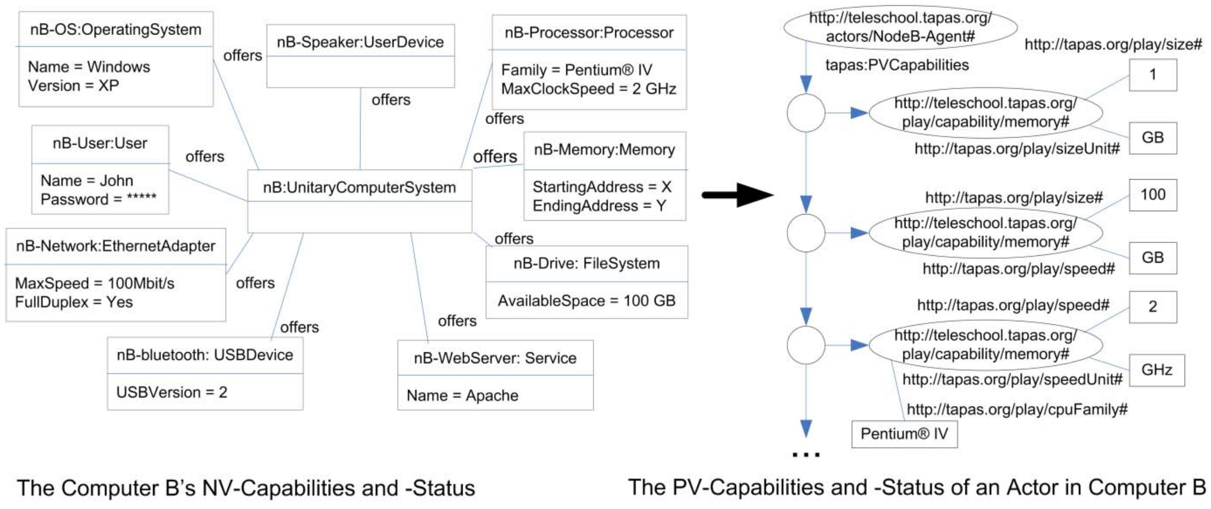

nB-OS:OperatingSystem
Name = Windows
Version = XP
offers
nB-Speaker:UserDevice
offers
nB-Processor:Processor
Family = Pentium® IV
MaxClockSpeed = 2 GHz
offers
nB-User:User
Name = John
Password = *****
offers
offers
nB-Memory:Memory
StartingAddress = X
EndingAddress = Y
nB:UnitaryComputerSystem
nB-Network:EthernetAdapter
MaxSpeed = 100Mbit/s
FullDuplex = Yes
offers
offers
nB-Drive: FileSystem
AvailableSpace = 100 GB
offers
offers
nB-bluetooth: USBDevice
USBVersion = 2
nB-WebServer: Service
Name = Apache
http://teleschool.tapas.org/actors/NodeB-Agent#
tapas:PVCapabilities
http://tapas.org/play/size#
1
http://teleschool.tapas.org/play/capability/memory#
GB
http://tapas.org/play/sizeUnit#
http://tapas.org/play/size#
100
http://teleschool.tapas.org/play/capability/memory#
GB
http://tapas.org/play/speed#
http://tapas.org/play/speed#
2
http://teleschool.tapas.org/play/capability/memory#
GHz
http://tapas.org/play/speedUnit#
http://tapas.org/play/cpuFamily#
Pentium® IV
...
The Computer B's NV-Capabilities and -Status
The PV-Capabilities and -Status of an Actor in Computer B

Figure 3: The transformation of NV-Capabilities and -Status of Computer B to PV-Capabilities and Status

One of the emerging standards in networking data model is *Common Information Model (CIM)* from Distributed Management Task Force (DMTF). The standard is based on the attempt to integrate all management information. On the basis of UML used in the model, CIM is an easy-to-use model with a lot of graphical modeling tools and Application Programming Interfaces (APIs) on many computing platforms. The model comes with sufficient pre-defined physical, logical and service entities with the possibility for extensions. The CIM serialization in the Extensible Markup Language (XML-CIM) inherits an important property of XML, the machine comprehensible. The example of modeling the NV-Capability and -Status in CIM is illustrated in Figure 3.

CIM provides only the basic semantic to each entity in the model. To cope with this limitation, we develop a framework that improves the expressiveness of CIM models by constructing XML Declarative Description (XDD)-based Configuration Rules with axioms, constraints and a reasoning mechanism to facilitate additional semantic to the model entities. XDD [11] is an expressive XML-based knowledge representation that works seamlessly with all models, syntax and ontologies by extending ordinary wellformed XML ground element with variables. Aagesen et al. gave an example of a dynamic configurable system in the Network View with CIM and XDD [2].

5.2 Play View Capability and Status Representation

In the Play View, an Actor inherits Capabilities and Status from a Node. These are Play View Capabilities and Status, abbreviated as PV-Capability and -Status. Unlike the NV-Capability and Status, the PV-Capability and Status are represented in a unified and platform independent representation: *Resource Description Framework* (RDF). Each PVCapability is an RDF resource with a *Uniformed Resource Identifier (URI)* that belongs specifically to a domain. An RDF statement is composed by *a triple consisting Subject, Verb, Object* that resembles an English sentence. Even though the RDF structure is simple, it provides sufficient constructs to describe PV-Capabilities and -Status. In addition, the standard ontology languages on the web such as the RDF Schema (RDFS) and the Web Ontology Language (OWL) are also based on RDF. This enhances the reasoning mechanism by the creation, manipulation and exchange of the ontology of the PV-Capability and -Status across Domains. Anutariya et al. proposed an RDF Declarative Description (RDD) framework, which shows the effectiveness of using the ontological language such as RDF, DAML+OIL and DAML-S with the Configuration Rules described in XDD. [4]. The RDD framework is the basic idea of employing PV-Capability and PVStatus in TAPAS.

Figure 3 shows the transformation from the Computer B’s NV-Capability represented in CIM instance to the equivalent PV-Capability in RDF. Due to the space limitation, the complete RDF diagram is not given.

5.3 Domain Oriented Representation

In TAPAS, a *Domain*, taken care by a single Director, is a part of a big system containing Nodes that are related in terms of location, functionality or service. The Internet is a good example of a Domain System. Computers on the Internet are divided into groups with unique domain names. UniCS is a domain-oriented representation, i.e. the semantic of Capability and the Status may vary from place to place, from time to time or from a specific situation to others. Since it is not possible to force a single semantic framework to all systems, a *Common Semantic* is used to resolve semantic conflicts in different Domains.

In Figure 4, each Domain is supplied with its own semantic with the relationship to the Common Semantic. The Common Semantic provides the relationship between identical Capabilities and Status from various Domains. It is used to resolve possible conflicts from Capabilities and Status with different semantic and thus make it possible to combine them. For example, the Smartphone concept is defined in a Domain as a mobile phone that requires Microsoft Windows Mobile Edition for Smartphone. This is because the Domain can only supply support software for this platform only. However, the case might not be the case in some Domains where the Smartphone Operating System can vary from Symbian, Microsoft Windows Mobile Edition or Linux. The concept of Domain applies to both NV-Capability and -Status and PV-Capability and -Status. Nonetheless, UniCS handles the NV-Capability and -Status differently.

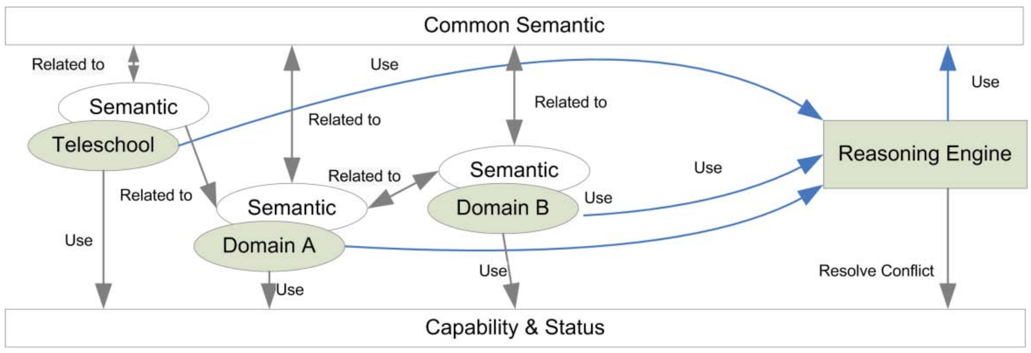


Figure 4: UniCS's ontological capability and status handling

The semantic of the NV-Capability or an NV-Status is mapped equivalently to one or more PV-Capabilities or -Status defined in the Domain. This reduces the complexity of handling the semantic differences of NV-Capabilities and -Status across domains, and thus provides a unified framework to the Capability and Status Management by using the PV-Capability and -Status. In addition, UniCS allows the flexible interoperability with other architectures that encode Capability and Status in different syntax and semantic. With Configuration Rules supplied, the Capability and Status representations in those architectures can be transformed equivalently into the UniCS's PV-Capability and -Status.

6. Using Play View Capability and Status

We have briefly shown how Capability and Status in TAPAS are represented in UniCS. Basically, UniCS separates NV-Capability and -Status from PV-Capability and -Status. Configuration Rules, which constitutes the Play in TAPAS, are created from PVCapabilities and -Status. The representation layer showing the representation used in UniCS is illustrated in Figure 5. We are now ready to use the PV-Capability and –Status to create Configuration Rules for a scenario in the teleschool environment.

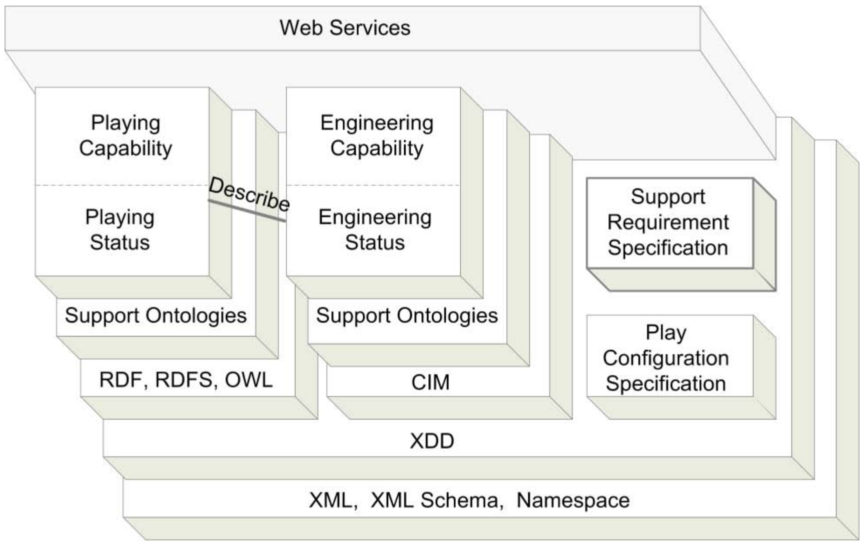
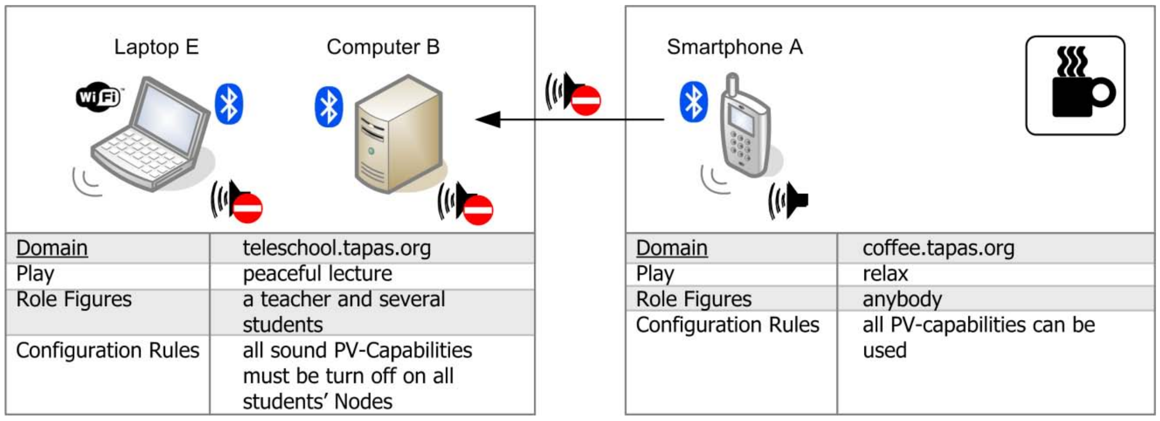


Figure 5: UniCS's Capability and Status Representation Layer (extended from [1])

Figure 6 reveals the facilities of a network classroom. Since it has been recently reported that many students don't turn off their mobile phones when they are in the classroom, the teleschool administrator is asked to construct a play in the TAPAS environment to automatically detect and request the smartphone Nodes to turn off the *Sound PV-Capability*. Fortunately, the students' smartphones are installed with the TAPAS Core Platform that provides support for the Play-based concepts.



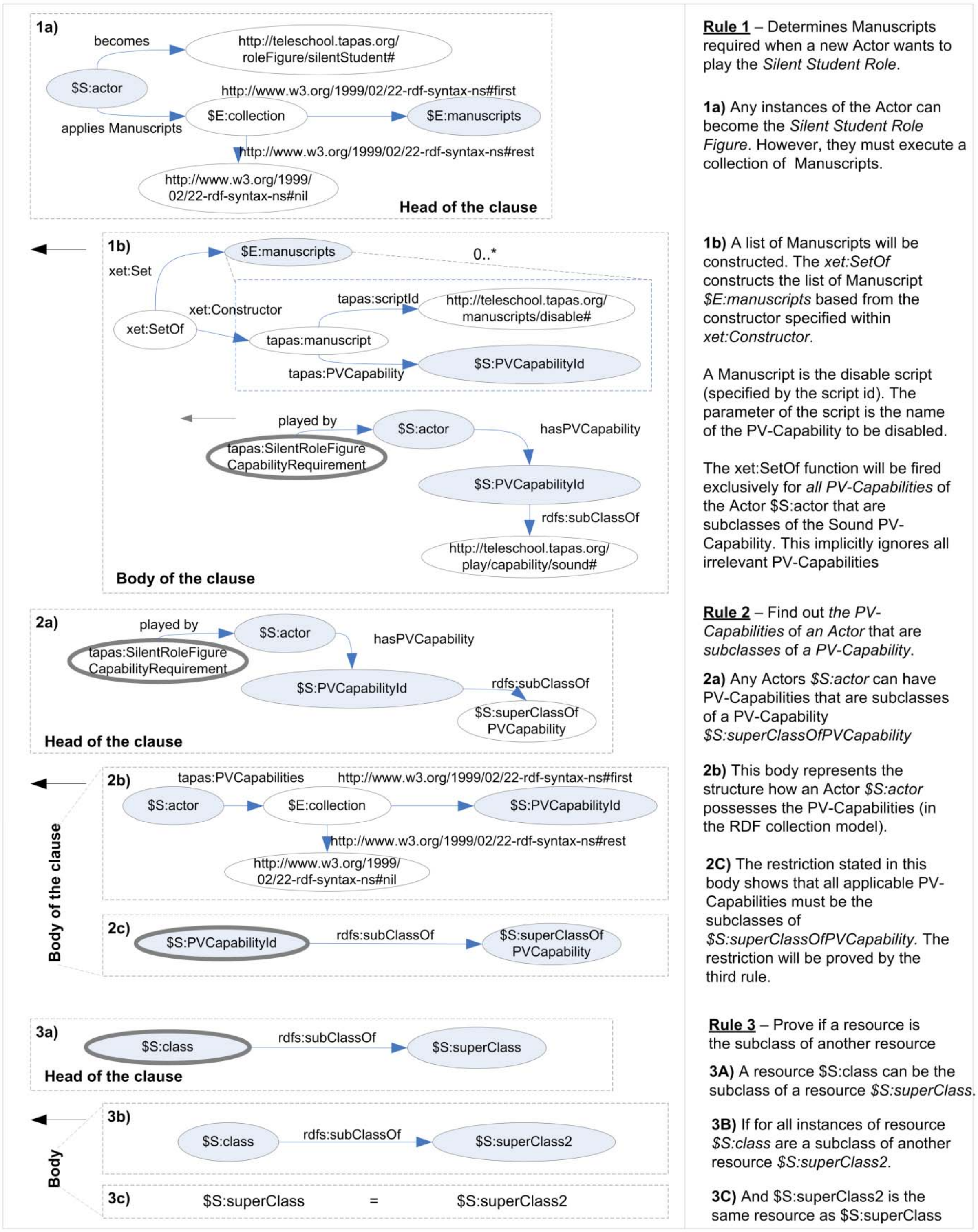
Domain	teleschool.tapas.org
Play	peaceful lecture
Role Figures	a teacher and several students
Configuration Rules	all sound PV-Capabilities must be turn off on all students' Nodes

Domain	coffee.tapas.org
Play	relax
Role Figures	anybody
Configuration Rules	all PV-capabilities can be used

Figure 6: The example of an adaptable network system in a network classroom

It is assumed that all PV-Capabilities that produce sound must be turned off separately. Therefore, the system administrator must create a Play to verify all PV-Capabilities of the Actors hosted in the students' Nodes, i.e. the smartphones. If Nodes have PV-Capabilities that are the *subclasses* of the *Sound PV-Capability*, the Director will send a *"Disable Manuscript"* to turn them off. Based on UniCS, three Configuration Rules are created as illustrated in Figure 7.

When the smartphone A, with an MP3player, a Loudspeaker and a Bluetooth PV-Capabilities, enters the classroom, two PV-Capabilities, the Loudspeaker and the MP3player, will be turned off. This is because they are the subclasses of the sound PVCapability. With the provided subclass-superclass ontology, the Director create a *Play Configuration* instructing an Actor in the smartphone A, smartphoneA-Agent, to turn off these PV-Capabilities with a Manuscript. The execution result is shown in Figure 8.


1a)
becomes
http://teleschool.tapas.org/roleFigure/silentStudent#
$S:actor
http://www.w3.org/1999/02/22-rdf-syntax-ns#first
$E:collection
$E:manuscripts
applies Manuscripts
http://www.w3.org/1999/02/22-rdf-syntax-ns#rest
http://www.w3.org/1999/02/22-rdf-syntax-ns#nil
Head of the clause
1b)
$E:manuscripts
0..*
xet:Set
xet:Constructor
tapas:scriptId
http://teleschool.tapas.org/manuscripts/disable#
xet:SetOf
tapas:manuscript
tapas:PVCapability
$S:PVCapabilityId
played by
$S:actor
hasPVCapability
tapas:SilentRoleFigureCapabilityRequirement
$S:PVCapabilityId
rdfs:subClassOf
http://teleschool.tapas.org/play/capability/sound#
Body of the clause
2a)
played by
$S:actor
hasPVCapability
tapas:SilentRoleFigureCapabilityRequirement
$S:PVCapabilityId
rdfs:subClassOf
$S:superClassOfPVCapability
Head of the clause
2b)
tapas:PVCapabilities
http://www.w3.org/1999/02/22-rdf-syntax-ns#first
$S:actor
$E:collection
$S:PVCapabilityId
http://www.w3.org/1999/02/22-rdf-syntax-ns#rest
http://www.w3.org/1999/02/22-rdf-syntax-ns#nil
Body of the clause
2c)
$S:PVCapabilityId
rdfs:subClassOf
$S:superClassOfPVCapability
3a)
$S:class
rdfs:subClassOf
$S:superClass
Head of the clause
3b)
$S:class
rdfs:subClassOf
$S:superClass2
Body
3c)
$S:superClass = $S:superClass2

Rule 1 – Determines Manuscripts required when a new Actor wants to play the *Silent Student Role*.

1a) Any instances of the Actor can become the *Silent Student Role Figure*. However, they must execute a collection of Manuscripts.

1b) A list of Manuscripts will be constructed. The *xet:SetOf* constructs the list of Manuscript *$E:manuscripts* based from the constructor specified within *xet:Constructor*.

A Manuscript is the disable script (specified by the script id). The parameter of the script is the name of the PV-Capability to be disabled.

The xet:SetOf function will be fired exclusively for *all PV-Capabilities* of the Actor $S:actor that are subclasses of the Sound PV-Capability. This implicitly ignores all irrelevant PV-Capabilities

Rule 2 – Find out *the PV-Capabilities* of *an Actor* that are *subclasses* of *a PV-Capability*.

2a) Any Actors *$S:actor* can have PV-Capabilities that are subclasses of a PV-Capability *$S:superClassOfPVCapability*

2b) This body represents the structure how an Actor *$S:actor* possesses the PV-Capabilities (in the RDF collection model).

2C) The restriction stated in this body shows that all applicable PV-Capabilities must be the subclasses of *$S:superClassOfPVCapability.* The restriction will be proved by the third rule.

Rule 3 – Prove if a resource is the subclass of another resource

3A) A resource $S:class can be the subclass of a resource *$S:superClass.*

3B) If for all instances of resource *$S:class* are a subclass of another resource *$S:superClass2.*

3C) And $S:superClass2 is the same resource as $S:superClass

Figure 7: The Play to turn-off all kinds of PV-Capabilities that produce sound.

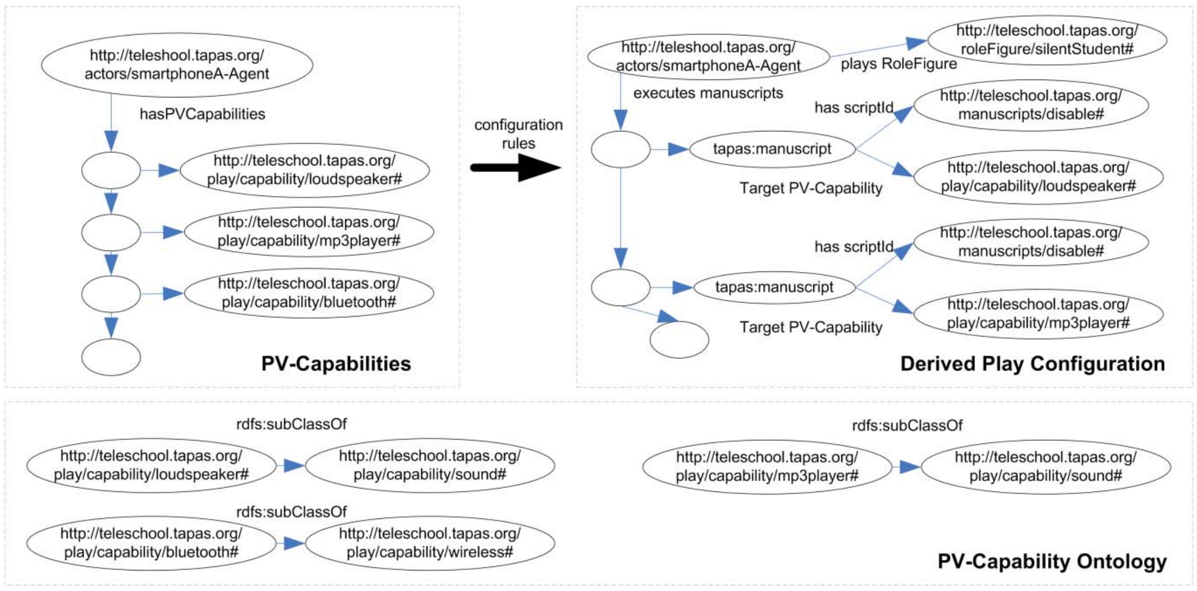

http://teleshool.tapas.org/ actors/smartphoneA-Agent
hasPVCapabilities
http://teleschool.tapas.org/ play/capability/loudspeaker#
http://teleschool.tapas.org/ play/capability/mp3player#
http://teleschool.tapas.org/ play/capability/bluetooth#
PV-Capabilities
configuration rules
http://teleshool.tapas.org/ actors/smartphoneA-Agent
plays RoleFigure
http://teleschool.tapas.org/ roleFigure/silentStudent#
executes manuscripts
has scriptId
http://teleschool.tapas.org/ manuscripts/disable#
tapas:manuscript
Target PV-Capability
http://teleschool.tapas.org/ play/capability/loudspeaker#
has scriptId
http://teleschool.tapas.org/ manuscripts/disable#
tapas:manuscript
Target PV-Capability
http://teleschool.tapas.org/ play/capability/mp3player#
Derived Play Configuration
rdfs:subClassOf
http://teleschool.tapas.org/ play/capability/loudspeaker#
http://teleschool.tapas.org/ play/capability/sound#
rdfs:subClassOf
http://teleschool.tapas.org/ play/capability/mp3player#
http://teleschool.tapas.org/ play/capability/sound#
rdfs:subClassOf
http://teleschool.tapas.org/ play/capability/bluetooth#
http://teleschool.tapas.org/ play/capability/wireless#
PV-Capability Ontology

Figure 8: The execution result of the Play in Figure 7 and the PV-Capabilities with supplied ontologies.

7 Conclusion

By using UniCS's PV-Capability and -Status, the semantic of the Capability and Status models of network systems can be enriched. As a result, a play-based network system, led by a Director, can orchestrate itself to behave properly when there are changes occurring. This reasoning mechanism serves as a principle of the *rearrangement flexibility* that is one of the three basic required properties for the *adaptable system*. Nevertheless, there are still more works to do regarding the rearrangement flexibility since the simple subclass-superclass relation is not expressive enough in all situations. We are working on the next version of UniCS to provide more ontological ingredients. The future integration with the RDD framework will enlarge the scope of the ontological framework in UniCS, and thus will make the ontological model suitable in most situations. In addition, the model must be combined with other works regarding failure Robustness and resource load awareness and Control in TAPAS. The final model will then serve as the true grounding for the Adaptable System.

8. Related Work

CIM is used in many projects to resolve the interoperability problems between systems. Examples are the Web-Based Enterprise Management (WBEM) and the Directory Enabled Networks (DEN) from DMTF [10]. However, the frameworks lack a mechanism to enable the dynamic behavior. The Automated Policy-based Resource Construction by Sahai et al. employs CIM as the underlying Capability model [8]. Nevertheless, the policies are constructed mainly by the defined constraints. The lack of expressive axioms limits the system from deriving new knowledge, which is unfortunately needed in adaptable environments.

There are several attempts trying to provide a unified framework to model syntax and semantics using XML and RDF. This is again emphasized by Tim Burners-Lee when he encouraged developers to start building RDF triples that contain ad-hoc ontologies in the WWW2004 conference. Patel-Schneider and Sim´eon employ an RDF mediator to allow XML dialects in the applications [7]. Although this is quite relevant to our work, their research focuses on providing the semantic reasoning to static models. Thus, it is not well-suited in the world of network application. The RDF-based system can also be found in the world of network management. Shen and Yang use the RDF to describe models created by the next generation structure of management information (SMIng) [9]. However, the work tends to focus mainly on the resource level, not Capability. Another attempt by Motik and Glavini´c to create model for querying RDF knowledge in the Agent Architecture is limited to the RDF and not applicable to other ontological languages [6].

References

[1] Finn Arve Aagesen, Chutiporn Anutariya, Mazen Malek Shiaa, and Bjarne E. Helvik. Support specification and selection in tapas. in *Proceedings of IFIP WG6.7 Workshop and Eunice Summer School on Adaptable Networks and Teleservices*, September 2002.

[2] Finn Arve Aagesen, Chutiporn Anutariya, Mazen Malek Shiaa, Bjarne E. Helvik, and Paramai Supadulchai. A dynamic configuration architecture. in *IEEE/IFIP Network Operations and Management Symposium NOMS'2004*, Seoul, South Korea, April 2004.

[3] Finn Arve Aagesen, Bjarne E. Helvik, Chutiporn Anutariya, and Mazen Malek Shiaa. On adaptable networking. in *Proceedings of the First International Conference on Information and Communication Technologies, ICT'2003*, Assumption University, Thailand, April 2003.

[4] Chutiporn Anutariya, Vilas Wuwongse, Kiyoshi Akama, and Ekawit Nantajeewarawat. Rdf declarative description (rdd): A language for metadata. *Journal of Digital Information*, 2(2), 2001.

[5] Shanshan Jiang and Finn Arve Aagesen. Xml-based dynamic service behaviour representation. in *NIK'2003*, Oslo, Norway, November 2003.

[6] Boris Motik and Vlado Glavini´c. Enabling agent architecture through an rdf query and inference engine. in *Proceedings of the 10th Mediterranean Electrotechnical Conference, MEleCon 2000*,volume 2, pages 762–765, Cyprus, 2000.

[7] Peter F. Patel-Schneider and J´erˆome Sim´eon. The yin/yang web: A unified model for xml syntax and rdf semantics. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):797–811, July/August 2003.

[8] Akhil Sahai, Sharad Singhal, Rajeev Joshi, and Vijay Machiraju. Automated policy-based resource construction in utility computing environments. in *IEEE/IFIP Network Operations and Management Symposium NOMS'2004*, Seoul, South Korea, April 2004.

[9] Jun Shen and Yun Yang. Rdf-based knowledge model for network management. in *Proceedings of the 8th IFIP/IEEE International Symposium on Integrated NetworkManagement (IM2003)*, Colorado Springs, CO, USA, March 2003. Kluwer Academic Publishers.

[10] Andrea Westerinen and Winston Bumpus. The continuing evolution of distributed systems management. *IEICE TRANS. INF & SYST.*, E86-D(11):2256–2261, November 2003.

[11] Vilas Wuwongse, Chutiporn Anutariya, Kiyoshi Akama, and Ekawit Nantajeewarawat. Xml declarative description (xdd): A language for the semantic web. *IEEE Intelligent Systems*, 16(3):54–65, May/June 2001.

Paper B

Configuration Management for Adaptable Service Systems

Finn Arve Aagesen, Paramai Supadulchai, Chutiporn Anutariya and
Malek Mazen Shiaa

In Proceedings of *IFIP International Conference on Metropolitan Area Network, Architecture, Protocols, Control and Management (MAN 2005),*

Ho Chi Minh City, Vietnam, 2005.

Configuration Management for an Adaptable Service System

Finn Arve Aagesen* — Paramai Supadulchai* —
Chutiporn Anutariya — Malek Mazen Shiaa***

** Department of Telematics*
Norwegian University of Science and Technology (NTNU)
N-7491 Trondheim, Norway
aagesen@item.ntnu.no
paramai@item.ntnu.no
malek@item.ntnu.no

*** Computer Science Program, Shinawatra University*
Pathumthani 12160, Thailand
chutiporn@shinawatra.ac.th

ABSTRACT: *Adaptable service systems are service systems that are capable of handling dynamic changes in both time and position related to users, capabilities, nodes and changed service requirements. The paper presents a formal framework for dynamic configuration and reconfiguration of services in TAPAS (Telematics Architecture for Play-based Adaptable Systems). The framework presented in this paper, provides representation and reasoning mechanisms for semantic description and matching of required and offered capabilities and status which are required by a particular service system. It employs CIM schema and recently developed languages for the Semantic Web—RDF, RDF Schema and DAML languages—in order to provide human-readable and machine-comprehensible descriptions of status, capabilities, system (re)configuration plans as well as the exchange of messages. It exploits XDD theory—an expressive XML rule-based, knowledge representation—to seamlessly unify such various languages into a single uniform formalism, hence allowing the integration, extraction, and reasoning with instances/objects of those different languages.*

KEYWORDS: *Autonomic Communication, Adaptable Systems, Dynamic configuration, Configuration management.*

1. Introduction

A network based service system consisting of services, service components and nodes is considered. A service is realised by the structural and behaviour arrangement of service components, which by their inter working provide a service in the role of a service provider to a service user. Service components are executed as software components in nodes, which are physical processing units such as servers, routers, switches and user terminals. User terminals can be phones, laptops, PCs and PDAs etc.

Network-based services have during more than one decade been is an important research topic. Example topics include TINA (Tele-communication Information Networking Architecture) (Inoue at al. 1999), Mobile Agents and Active and Programmable Networks (Bieszczad et al. 1998) (Raza et al. 2004) (Tennenhouse et al. 1997). Focus has been on service architecture solutions that give flexibility and efficiency in the definition, deployment and execution of the services. This focus is now slightly changing into focus on adaptability and evolution of such services. Traditionally, the nodes as well as the service components have a predefined functionality. Concerning nodes as well as software engineering principles, changes are taking place. Nodes are getting more generic. A modern node may offer IP telephony and can have an MP3 player, video camera, storage etc. In the same way, the software components are getting more generic. From being static components, the software components can be 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*. The name actor is chosen because of the analogy with the actor in the theatre, which is able to play different roles in different plays.

We are entering a generative era, which gives a high degree of flexibility. To utilise the generative potential, the attributes of services, service components, software components and nodes must be appropriately formalised, stored and made available. There must also be generative platform functionality that utilises this generative data. Generative data and functionality apply to the ordinary service functionality, but also to the service management functionality. As a first step towards this formalisation, the concepts *capability* and *status* are introduced.

A capability is an inherent property of a node or a user, which defines the ability to do something. A capability in a node is a feature available to implement services. An actor executes a program. However, this program may need capabilities in the node. A capability of a user is the feature that makes the user capable of using services. Capabilities can be classified into:

- Resources: physical hardware components with finite capacity,
- Functions: pure software or combined software/hardware components, which perform particular tasks,
- Data: just data, which interpretation, validity and life span of which depend on the context of the usage.

Resource capability examples are processing, storage and communication resources e.g., CPU, hard disk and transmission channels, standard equipment e.g., printers and media handling devices and special equipment e.g., encryption devices. Function capabilities are functions related to the use of hardware resources, such as encryption, and special programs or library functions available for general use. Data capability examples are user login and access rights

Status is a measure for the situation in a system with respect to the number of active entities, the traffic situation and the Quality of Services (QoS) etc. Status reflects an instantaneous state of the system. It can comprise observable counting measures, or calculated QoS measures.

The work presented in this paper has been related to the *Telematics Architecture for Play-based Adaptable System* (TAPAS) (Aagesen et al. 1999, 2001, 2002, 2003). The TAPAS *computing architecture*, to be more detailed explained in Section 3, defines a service system by a play. A play consists of several actors, constituting role figures by playing roles. A role figure is realised in an executing environment in a node, and is utilising capabilities, which are inherent properties of the node. A role can have specific requirements to capabilities and status. Due to the dynamic availability of nodes in the network as well as changes in their capabilities and status, it is desirable that configuration of services is done dynamically. *Configuration management* is the optimisation of service systems initial configuration and reconfigurations with respect to capabilities and status. This is the focus of this paper.

Section 2 discusses related works. Section 3 gives a brief outline of TAPAS architecture. Section 4 proposes a dynamic configuration framework. Its data model and reasoning mechanism are presented in Section 5. Section 6 demonstrates a practical application of the framework together with the reasoning mechanism. Section 7 concludes and outlines further research direction.

2. Related work

Several configuration management and adaptable architectures have been proposed so far (Bakour et al. 2004) (Cohen et al. 2004) (D'Antonio et al. 2004) (Keller et al. 2004) (Sahai et al. 2004) (Solarski et al. 2004). Nevertheless, they are most likely the architectures to handle a specific task, which either can be the *service creation and deployment functionality* (Bakour et al. 2004) (Cohen et al. 2004) (Keller et al. 2004) (Solarski et al. 2004) or the *network and resource management functionality* (D'Antonio et al. 2004) (Sahai et al. 2004). Our architecture is intended to provide a configuration management for any adaptable system that provides the functionality for both service creation and deployment network and resource management. This diversity comes from the use of *XML Declarative Description (XDD)*, a generic knowledge representation. XDD provides a single uniform formalism to create knowledge that incorporates various capability and status representations as well as service behavior representations. Moreover, the ability to effectively handle different kinds of event messages, which are well categorised in an ontology instance, and the underlying reasoning mechanism guarantee that an event happening in the system will be handled by rule-based procedures that can apply to them. The reasoning mechanism transforms an event message *equivalently* with the supplied configuration rules until a proper procedure to handle the event is obtained. The transformation preserves all the semantic in a service system (Wuwongse et al. 2001).

3. TAPAS architecture

TAPAS intends to be an architecture that gives 1) rearrangement flexibility, 2) failure robustness and survivability, and 3) resource load awareness and control (Aagesen et al. 2003). The TAPAS architecture is separated into a *computing* architecture and a *system management* architecture as follows:

- The computing architecture is a generic architecture for the modeling of any service software components
- The system management architecture is the structure of services and service management components.

These architectures are not independent and can, to some extent, also be seen as architectures at different abstraction layers. The system management architecture, however, has focus on the functionality independent of implementation, and the computing Architecture has focus on the modeling of functionality with respect to implementation, but independent of the nature of the functionality. The nature of the computing as well as system architecture is described briefly in the following.

3.1. Computing architecture

TAPAS computing architecture has three layers: the *service view*, the *play view* and the *network view* as shown in Figure 1. The service view concepts are rather generic and should be consistent with any service management architecture. Likewise, the network view concepts are generic and should be consistent with any corresponding network architecture, with exception of the *core platform*, which is a specific platform supporting the play view concepts. The network view concepts are the basis for implementing the play view concepts, which again are the basis for implementing the service view concepts. In the other way around, the service view concepts are mapped into the play view concepts, which again are mapped into the network view concepts.

The play view intends to be a basis for designing functionality that can meet the rearrangement, the robustness, the survivability, the QoS awareness and resource control properties. The play view concepts are seemingly rearrangement flexibility oriented. The capability and status concepts, however, also give a basis for the further design of the robustness, the survivability, the QoS awareness and resource control properties.

In the network view, *nodes* are typically network processing units such as mobile phone, desktop computer, laptop, printer and router that possess particular capabilities. Nodes are installed with *core platform*. Core platform supports basic communication infrastructure between nodes. At a specific time point, status is the state of a system with respect to the number of active entities, traffic situation and QoS etc.

The play view architecture is founded on a *theater metaphor*. The TAPAS actor is a generic software component consistent with the actor definition given in Section 1. However, the TAPAS actor is specialised as follows. Actors perform *roles* according to predefined *manuscripts*, and a *director* manages their performance. Actors are software components in the nodes that can download manuscripts. An actor will constitute a *role figure* by behaving according to a manuscript that defines the functional behavior of that particular role in a *play*. A *role Session* is a projection of the behavior of a role figure with respect to one of its interacting role figures. Actors in TAPAS can be moved transparently between nodes and the role sessions between them can be re-instantiated automatically (Shiaa 2004).

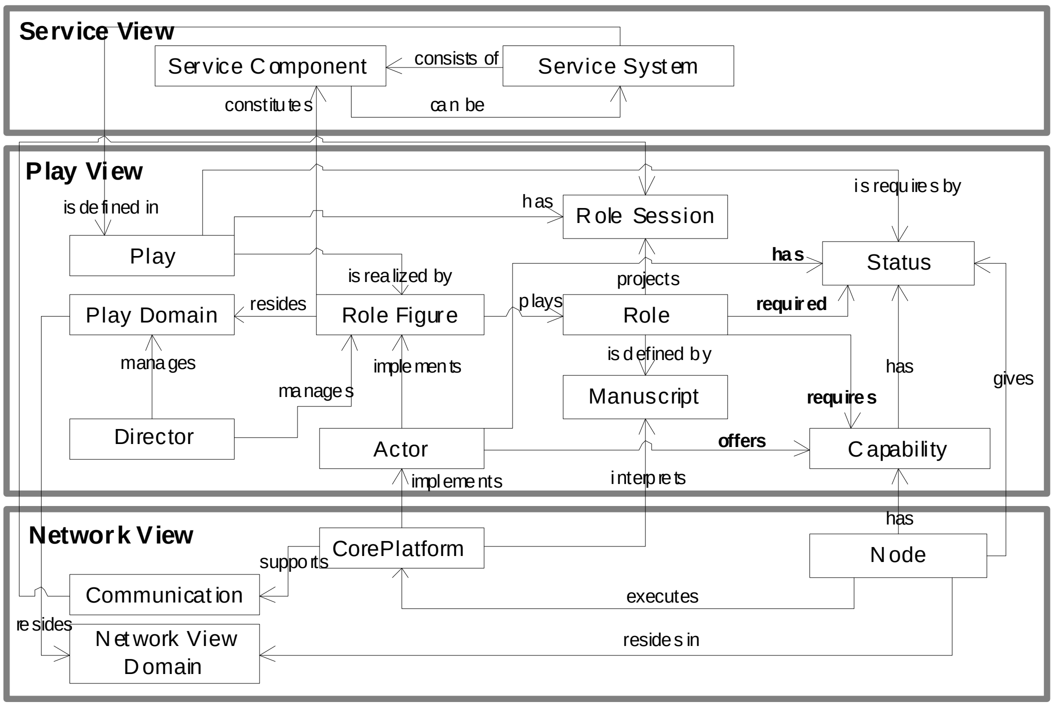


Figure 1. The Simplified TAPAS Computing Architecture

A director is an actor with supervisory status regarding other actors. A director also represents a play view domain, which is a set of nodes, which actors are supervised by a single director. The director chooses a fitting actor for a certain role figure. For this task the director requests help from the service management functionality defined in Section 3.2.

A *service system* is defined by a *play*. A play consists of several actors playing different roles, each possibly having different requirements on capabilities and Status. An actor will constitute a role figure, based on the role defined by a manuscript. 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 of the actor is executing (Aagesen et al. 2003).

3.2. System management architecture

The main functionality components of the system management architecture are illustrated in Figure 2. To fulfill the failure robustness and survivability requirements, the architecture must be dependable and distributed, this means replication of resources and functionality. The dependability aspect is beyond the scope of this paper, and the various functionality components will be defined as being part of a centralised architecture. The *Primary Service Providing Functionality* comprises the ordinary services offered to Users. In addition, the following functionality components are defined:

- *Service Management*: Definition of new services, deployment and invocation of services and Service Components

- *Capability and Status Management*: Registration, de-registration, update, transform and provide access to capabilities and status repository.
- *Configuration Management*: Optimisation of service systems initial configuration and re-configuration with respect to the capabilities and QoS.
- *Mobility Management*: The handling of various mobility types.

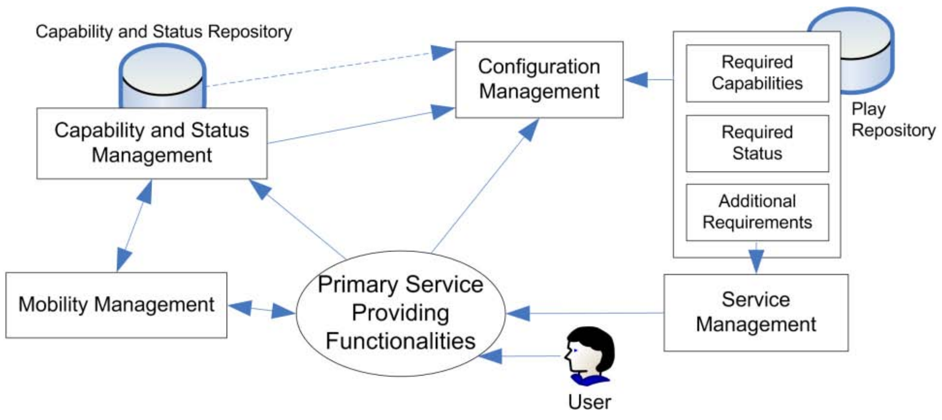


Figure 2. System management architecture functionality components

The functionality of these functionality components is constituted by the cooperation of role figures. Each of these functionality components has one *dedicated role figure*, which constitutes the main role figure within the functionality component, acting as the visible interface to the other functionality components. This main role figure is denoted as the manager. In this paper, a functionality component is considered to consist of the manager only. The functionality components defined above are accordingly replaced by the *Service Manager, the Capability and Status Manager, the Configuration Manager and the Mobility Manager, respectively.* This paper has focus on Configuration Management and the Configuration Manager. Aspects of the other functionality components without relevance to Configuration Management are beyond the scope of this paper.

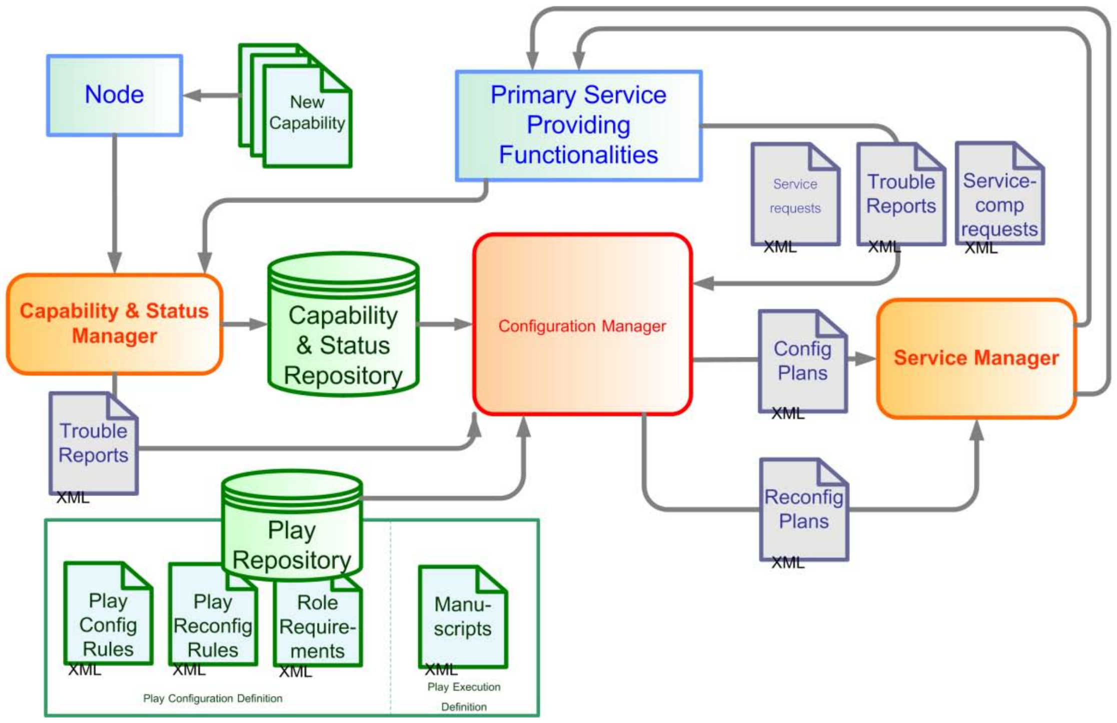


Figure 3. Architectural framework for dynamic configuration

4. Dynamic configuration framework

Figure 3 describes an architectural framework for dynamic configuration and reconfiguration of services. The main entities are the Configuration Manager, the Capability & Status Repository, the Play Repository, the Capability and Status Manager, and the Service Manager.

The Configuration Manager (CM) is responsible for:

- Generation of appropriate configurations for composing new services to be installed in a system: When there arises a request for installing a new service (i.e., a service request), the CM fetches a corresponding play definition and retrieves the system capabilities and status from the Play Repository and the Capability and Status Repository, respectively. Valid con-figurations for such a service are generated and analysed, and an appropriate configuration will be selected based on the specified selection criteria such as system performance and QoS, user preferences and cost. The selected configuration, defining which nodes in the system should execute actors constituting certain roles, will be forwarded to and executed by the Service Installer.

- Determination of a location for executing a particular role: In the running system, a request for instantiation of a particular service component (i.e., a service component request) may arise. In response to such a request, the CM dynamically determines the best location (node) for its installation based on the current system configuration, available capabilities and status as well as the component's requirements. It then notifies the Service Installer to load a corresponding manuscript from the Play Repository and instantiate it on the suggested node.

- Determination of reconfiguration schemes for dynamic reconfiguration of existing service systems: Upon the receipt of a trouble report indicating a problem in a running system, the CM analyses the problem, fetches related information from the Capability and Status Repository and the Play Repository, and computes a service reconfiguration plan to be executed by the Service Manager. Possible plans include actor relocation, re-initialisation, load balance and distribution. Selection of an appropriate plan depends on the defined reconfiguration rules as well as the nature of a problem (e.g., whether it is hardware or software failure, significant or ignorable).

Capability & Status Repository (*CSRep*) stores specifications of capabilities offered by components in a system and maintains information reflecting the situation and status of the system at a particular time. Such status information can be certain environment conditions, observable values of the current QoS characteristics as well as their calculated measures, which will be analysed by the Configuration Manager when computing (re)configuration plans for the system.

Play Repository is a collection of *play configuration definitions* and *play execution definitions*, which respectively define requirements and functional behaviour of a corresponding service system. A play configuration definition is an aggregation of the three specifications:

- *Role requirements* identify, for each role, its requirements on available capabilities/status.

- *Play configuration rules* describe system configuration rules and constraints which must always be maintained, such as the maximum number of roles allowed to install at a specific node in order to avoid an overload situation, the desired or acceptable QoS levels of the system, optional and mandatory constraints as well as conflict handling and priority information.

- *Play reconfiguration rules* define application-specific reconfiguration policies for handling significant reconfiguration-related events, such as a service component failure, a decrease in system QoS and resource unavailability. With application-specific reconfiguration rules, the system can perform appropriate actions to handle a problem in a running system.

A play execution definition consists of *manuscripts* which define the entire functional behaviour of each role in a play and include not only its internal behaviour, but also the interactions and cooperation with other roles. Note that role specifications and manuscripts define two different aspects of roles in a play. The former describes the metadata of each role and the latter models its functional behaviour in terms of EFSM (Extended Finite State Machine). Hence, they provide information of “what the role is” and “how it is realised”, respectively.

Capability and Status Manager monitors system capabilities/status and maintains the Capability and Status Repository. It also listens to certain events indicating changes to the system and its environment, which would prevent the system from getting the desired level of services. In response to such events, it notifies the Configuration Manager for further proper reactions in order to keep the system functioning with an acceptable QoS level. Capability and Status Manager is also responsible for installation and de-installation of capability components. When a new node with not-yet-installed capability components is plugged into the system, these components will be installed according to certain well-defined procedures, and their capabilities will be registered as part of the system. Similarly, de-installation of a component requires an execution of certain procedures and deregistration of the component’s capabilities.

Service Manager installs a service into the system by creating corresponding actors for execution of certain roles according to an obtained play configuration generated by the CM. Allocation of capabilities as well as instantiation of a manuscript for each role are also performed by this entity. The Service Manager also initiates and performs reconfiguration of a service system based on an obtained plan.

5. Data model

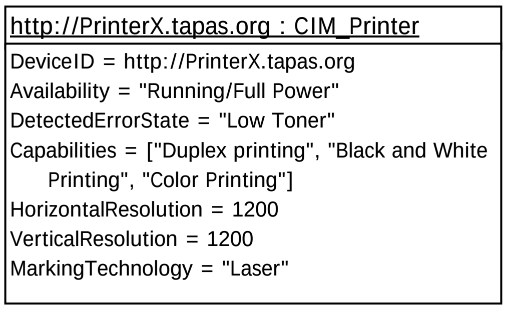
This section presents an XML declarative approach to the representation of dynamic configuration data model. It elaborates a machine-comprehensible description of each data/message element of the configuration framework proposed in Section 2 by employment of recently developed standard languages for the Semantic Web (Berners-Lee et al. 1999) and network management. Firstly, a mechanism for semantic description of system status and capabilities is discussed, followed by the formalisation of exchanging messages and play configuration definitions.

5.1. Capability and status specification

The developed framework proposes the use of standard XML-based metadata and ontology languages for modelling and providing semantic description of system capabilities and status. RDF (Brickley et al. 2004) (Lassila et al. 1999), which is a W3C recommended metadata language and its extensions (e.g., DAML (Hendler et al. 2000) and OWL (McGuinness et al. 2004)) appear to meet this language need. However, so far a standard, common ontological schema for describing network management resources in such languages does not yet exist. Therefore, the framework employs and extends CIM schema (Westerinen et al. 2000), developed by DMTF (Distributed Management Task Force) for representing capabilities and status. CIM is a fundamental, yet comprehensive object-oriented schema, both with respect to classifications and associations of objects, for describing network management information in a standard MOF (Managed Object Format) and XML format. In CIM model, the notions of capabilities and status are represented together as parts of an object’s properties. Figure 4 gives an example of a CIM instance, represented in both UML graphical notation and its

XML serialisation; it describes capabilities, status and certain operational attributes of a printer.

Based on these modelling concepts, the Capability and Status Repository is then represented as a collection of CIM instances which describe the available capabilities and status of the plug-and-play system. Note that to conform to W3C standards, CIM schemas and instances encoded in RDF can also be used in the proposed framework. However, in the open, heterogeneous environment, it is impossible to assume that every component/system will solely employ CIM model for semantic description of its capabilities and status. Thus, with this concern, research on integration of different capability/status ontologies is also part of the TAPAS project.



http://PrinterX.tapas.org : CIM_Printer

DeviceID = http://PrinterX.tapas.org
Availability = "Running/Full Power"
DetectedErrorState = "Low Toner"
Capabilities = ["Duplex printing", "Black and White Printing", "Color Printing"]
HorizontalResolution = 1200
VerticalResolution = 1200
MarkingTechnology = "Laser"

Note: The instance being described is an object of the class *CIM_Printer*. Its *DeviceID* property, used for uniquely identifying a device, states that the instance is identified by *http://PrinterX.tapas.org*. Its availability status is Running/Full Power with a low toner error state. Its printing capabilities include duplex, black & white, and colour printing. The horizontal and vertical resolutions are 1200 pixels per inch (this unit of measurement is predefined by CIM schema). Available character sets for the output are utf-8, us-ascii and iso-8859-1. Its marking technology is laser.

(a) UML graphical representation.

```
<INSTANCE ClassName="CIM_Printer">
  <PROPERTY NAME="DeviceID">
    <VALUE>http://PrinterX.tapas.org</VALUE>
  </PROPERTY>
  <PROPERTY NAME="Availability">
    <VALUE>Running/Full Power</VALUE>
  </PROPERTY>
  <PROPERTY NAME="DetectedErrorState">
    <VALUE>Low Toner</VALUE>
  </PROPERTY>
  <PROPERTY.ARRAY NAME="Capabilities">
    <VALUE.ARRAY>
      <VALUE>Duplex Printing</VALUE>
      <VALUE>Black and White Printing</VALUE>
      <VALUE>Color Printing</VALUE>
    </VALUE.ARRAY>
  </PROPERTY.ARRAY>
  <PROPERTY NAME="HorizontalResolution">
    <VALUE>1200</VALUE>
  </PROPERTY>
  <PROPERTY NAME="VerticalResolution">
    <VALUE>1200</VALUE>
  </PROPERTY>
  <PROPERTY NAME="MarkingTechnology">
    <VALUE> Laser</VALUE>
  </PROPERTY>
</INSTANCE>
```

(b) XML serialisation.

Figure 4. A CIM instance describing a printer device

5.2. Message specification

Because CIM does not provide means for representing various types of messages required by the developed architecture, RDF and DAML languages are exploited. Figure 5 illustrates various types of messages and gives their primitive attributes. Basically, each message carries its URI (Universal Resource Identifier), information of the actor who sends the message (i.e., the sender) and the date/time of composing it. A sender's information includes its URI, the installing location and the playing role. Other message attributes can also be encoded depending on the purpose of the message.

In the architecture, messages (cf. Figure 5) are classified into two main types: *requests* and *trouble reports*. Requests are further divided into: *service request* and *service component request*. The former is a request for installation and execution of a particular service system which has not yet been installed while the latter is a request for instantiation of a particular service component in a running service system. Figure 6 gives examples of both types of requests.

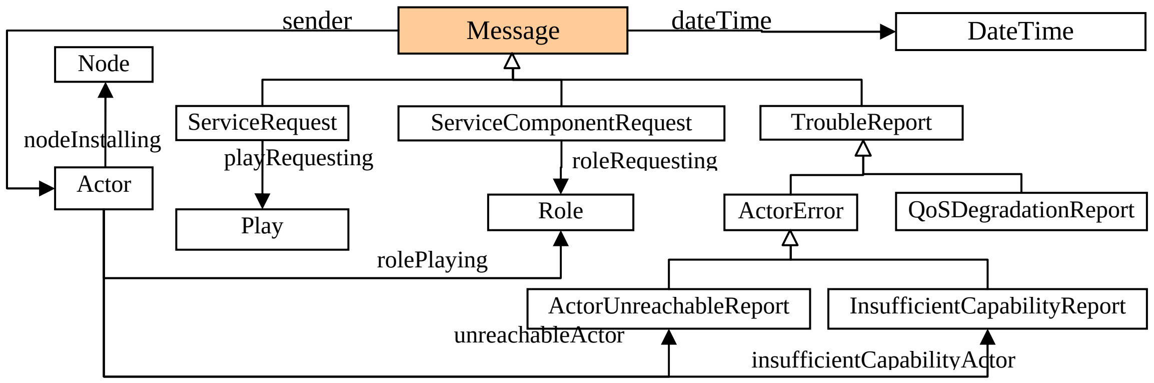


Figure 5. Message specification modeling

According to Figure 5, trouble reports are classified into: *QoS degradation report* and *actor error report*, which are used for notifying the CM when a QoS-sensitive service system encounters a decrease in its QoS to an unsatisfactory level and when an actor-involving problem occurs, respectively. Two types of actor error reports are: (i) *actor unreachable*: when an actor in a running system wants to communicate and cooperate with another existing actor which constitutes a particular role but is somehow unreachable or not responding, the former generates a trouble report indicating such a problem to CM; (ii) *insufficient capability*: an actor sends this type of trouble reports to the Configuration Manager if the node where it is installing and running has insufficient capabilities (such as insufficient disk space, memory or CPU speed) to satisfy its capability requirements.

5.3. Play configuration definition

This section elaborates the formalisation of the *play configuration definitions* by means of *XML Declarative Description (XDD)* language (Wuwongse et al. 2001, 2003). Recall that a play configuration definition comprises the following three parts: *role requirements*, *play configuration rules* and *reconfiguration rules*.

5.3.1. Role requirement

Capability and status requirement specification of a certain role in a play is expressed by an *XML clause*. Its head specifies the role to be played, and its body describes the demanded capabilities and status of a node for fulfilling such a role. Recall that 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. Given a clause representing a role requirement specification, one can derive a list of available nodes in the network, which are capable of performing such a role. By means of CIM hierarchical schema, matching of the required capabilities and status with the offered capability and status will not only be based on exact match, but will also include a notion of reasoning through this generalisation-specialisation hierarchy. For example, if a certain role demands a computer system with a modem, knowing that PC is a subclass of computer system, and unimodem, ISDN, ADSL and cable modems are subclasses of modem, then one can derive that any PC having one of these variety types of modems has sufficient capabilities to fulfil such a requirement. Ranking of available nodes according to how closely their capabilities match with the requirements is also expressed as XML clauses. Moreover, in case there are multiple nodes satisfying the defined requirements, specification of selection preferences is also permissible by appropriate formulation of conditions in the clause's body.

```
<ServiceRequest  rdf:about="http://tapas.org/msg01">
  <sender>
    <Actor rdf:about="http://tapas.org/actorA">
      <rolePlaying rdf:resource="http://tapas.org/roleR"/>
      <nodeInstalling
rdf:resource="http://comp1.tapas.org"/>
    </Actor>
  </sender>
  <dateTime>14/10/2002 GMT 14:10:00</dateTime>
  <playRequesting
rdf:resource="http://tapas.org/IPM_1.0"/>
</ServiceRequest>
```

(a) ServiceRequest.

```
<ServiceComponentRequest
rdf:about="http://tapas.org/msg02">
   <sender>
      <Actor rdf:about="http://tapas.org/actorB">
        <rolePlaying
rdf:resource="http://tapas.org/printClient"/>
        <nodeInstalling rdf:resource="http://comp1.tapas.org" />
      </Actor>
   </sender>
   <dateTime>14/10/2002 GMT 15:10:00</dateTime>
   <roleRequesting
rdf:resource="http://tapas.org/GraphicMaster"/>
</ServiceComponenetRequest>
```

(b) ServiceComponentRequest.

Figure 6. A service request and a service component request example.

5.3.2. Play configuration rules

Play configuration rules are represented as a set of XML clauses. Their heads identify components of the play, while their bodies describe the configuration, composition and dependency conditions. For example, consider a play of a distance-learning application consisting of the three roles: lecture-server, teacher and student together with a restriction that the actors playing the teacher and student roles must not be initialised at the same node. This condition is formalised as a clause with its head specifying these three roles of the play and its body constraining that the nodes for executing actors playing the teacher and client roles must not coincide.

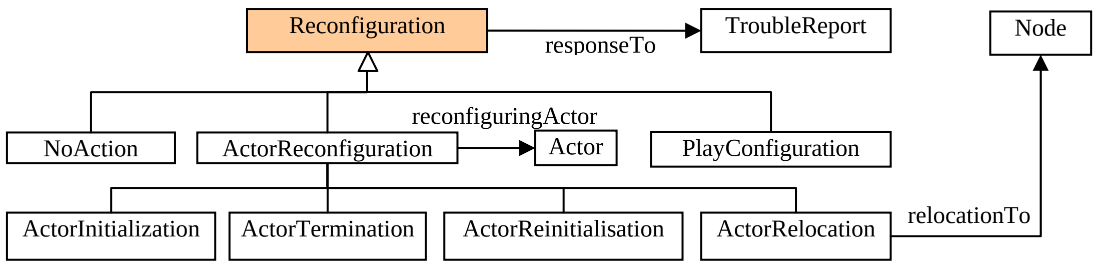


Figure 7. Reconfiguration types

5.3.3. Play reconfiguration rules

Instead of providing merely a general reconfiguration mechanism, which is applicable to any trouble encountered in an application, the developed framework additionally facilitates means for definition of application-specific *play reconfiguration rules*. Such rules let different applications encode their individual, customised reconfiguration policies, and hence allowing them to handle the same trouble in different but application-specific manners.

Each time when CM receives a trouble report, it will find if there is a reconfiguration rule specifically defined for handling the given trouble or not. In case that such a rule exists, CM will generate an appropriate system reconfiguration plan according to that rule. Otherwise, the default reconfiguration, i.e., to relocate actors that are involving in the problem, will be taken place. Figure 7 defines the possible types of reconfigurations: No Action, Play Reconfiguration and Actor Reconfiguration

No Action: System developers may decide to disregard and perform no action for certain types of troubles. For instance, one may define that all actor-error reports, which involve some particular low-priority roles and are submitted during 1AM-6AM, will be ignored.

Play Reconfiguration: The whole running service system, defined by the specified play and consisting of multiple cooperating actors, will be reconfigured. The best node for executing each actor will be re-determined and the actor will be relocated to that new location.

Actor Reconfiguration: This requires reconfiguration of some particular service components constituted by corresponding actors in a system, and can be further classified into Actor Initialisation, Actor Termination, Actor Re-initialisation, Actor Relocation.

- *Actor Initialisation*: The action is decomposed into (i) the instantiation of a new actor at a specified node, (ii) the installation of the manuscript defining the actor behaviour which corresponds to the role to be played, (iii) the execution of the actor's operation according to the installed manuscript.

- *Actor Termination*: The specified actor will be terminated and the resources allocated to and consumed by that actor will be freed.

- *Actor Re-initialisation*: The specified actor will be terminated and re-initialised at the same node.

- *Actor Relocation*: It involves moving of an actor currently executing at one node to another. In general, this reconfiguration is carried out when an actor has insufficient capabilities to execute its functions; thus, to proceed with its operation, the actor must be relocated to a node with sufficient capabilities. The references (Shiaa et al. 2002, 2004) have already discussed how *actor mobility* is realised in TAPAS.

A reconfiguration rule is formalised as an XML clause. Its head describes the reconfiguration action to be implemented, and its body represents the types, conditions and details of troubles upon which the described reconfiguration will be performed. For a given trouble report, there may exist more than one reconfiguration rule applicable to handle it. In such a case, rule prioritisation information is needed.

Table 1 summarises the developed model for the TAPAS dynamic configuration framework. It uniformly formalises the descriptions, classifications and constraints within a single representation scheme, i.e., XDD.

Table 1: Dynamic configuration data model

Modelling Components		Expressed as
Capability & Status Repository		
• Capability and Status Ontologies		CIM/XML schemas
• Capability and Status Specifications (Instances)		CIM/XML instances
Message Specifications		
• Service Requests		RDF/XML instances
• Service Component Requests		RDF/XML instances
• Trouble Reports		RDF/XML instances
Play Repository		
• Role Requirements		XML clauses
• Play Configuration Rules		XML clauses
• Play Reconfiguration Rules		XML clauses
A dynamic configuration system (capabilities and status + message spec. + play definitions)	Modelled as $\Rightarrow$	An XDD description comprising XML instances + XML clauses

** The semantics of the description yields appropriate system (re)configurations according to the current system capabilities and status, the defined play configuration and reconfiguration rules as well as the given requests and trouble reports.

6. Demonstration: Intelligent Printing Management System

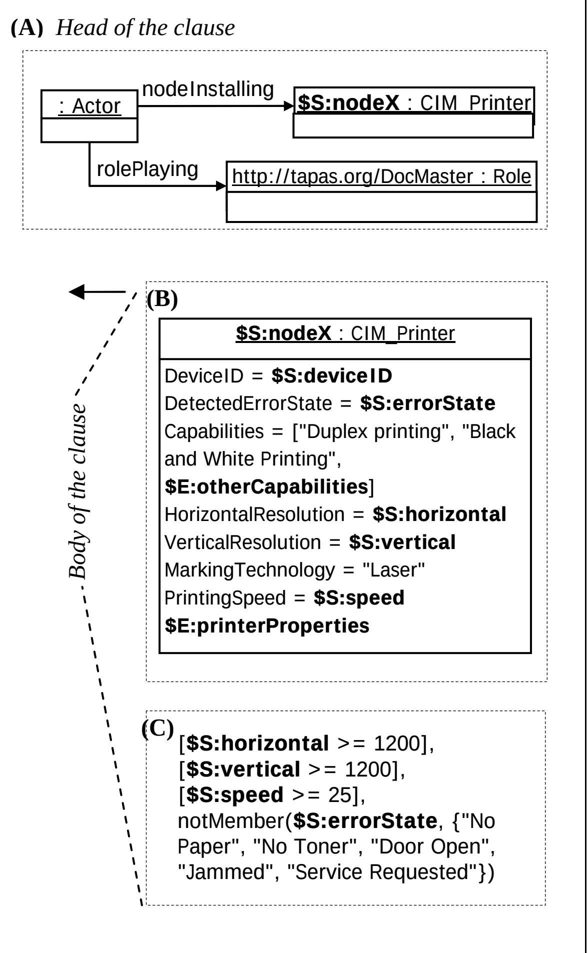
It is seen from the presented dynamic configuration architecture that CM is the primary entity which dynamically computes appropriate service (re)configuration plans by reasoning about the current system's capabilities and status, the defined role requirements, play configuration constraints and reconfiguration rules as well as the given requests and trouble reports. The prototype reasoning system for CM has already been developed by means of *XET* (*XML Equivalent Transformation*) (Anutariya et al. 2002)—a declarative style XML rule-based programming language and inference engine which can directly operate and reason about XML elements and XML clauses. Here, employment of the developed architecture and the reasoning engine to, respectively, model and implement an *Intelligent Printing Management* (*IPM*) system is demonstrated along with a simple application scenario assuming the four different roles:

- *DocMaster*: a print server role for printing black & white documents.
- *GraphicMaster*: a print server role for handling colour and graphic documents.
- *IPMManager*: a role responsible for controlling and distributing print jobs to appropriate printer roles, depending on the job attributes, the current queues of each printer and the job owner privilege information. It queries and finds an appropriate print server role for executing a given job. When there exists more than one print server role capable of handling the job, a preferred one will be selected.

- *PrintClient*: an application program to which users use for sending print jobs to IPM Manager.

Note that one printer can constitute more than one print server role, and a print server role can be realised by one or more physical printers. For instance, a high-speed, laser, colour printer may be configured to play both DocMaster and GraphicMaster roles, while the DocMaster role can be additionally realised by another black-and-white, laser printer. Moreover, in a real application scenario, there could be more varieties and more complicated types of print server roles for which different groups of users have different access controls.

Upon the system starts up, IPMManager and print server objects will be installed and configured. These objects receive their actual behaviour in manuscripts included in a general play definition which consists also of capability and status requirement definition. Clients can be plugged in later on at any possible node running the required TAPAS support. What is important from the point of view of dynamic configuration is the reasoning about these play and role requirements when installing specific roles at specific nodes.


(A) Head of the clause
: Actor
nodeInstalling
$S:nodeX : CIM_Printer
rolePlaying
http://tapas.org/DocMaster : Role
(B)
Body of the clause
$S:nodeX : CIM_Printer
DeviceID = $S:deviceID
DetectedErrorState = $S:errorState
Capabilities = ["Duplex printing", "Black and White Printing", $E:otherCapabilities]
HorizontalResolution = $S:horizontal
VerticalResolution = $S:vertical
MarkingTechnology = "Laser"
PrintingSpeed = $S:speed
$E:printerProperties
(C)
[$S:horizontal >= 1200],
[$S:vertical >= 1200],
[$S:speed >= 25],
notMember($S:errorState, {"No Paper", "No Toner", "Door Open", "Jammed", "Service Requested"})

(a) Graphical notation.

```
<Actor>
  <rolePlaying  rdf:resource="http://tapas.org/DocMaster"/>
  <nodeInstalling  rdf:resource=$S:nodeX />
</Actor>
   ← <INSTANCE  ClassName="CIM_Printer">
        <PROPERTY  NAME="DeviceID">
          <VALUE>$S:deviceID</VALUE>
        </PROPERTY>
        <PROPERTY  NAME="DetectedErrorState">
          <VALUE>$S:errorState</VALUE>
        </PROPERTY>
        <PROPERTY.ARRAY  NAME="Capabilities">
          <VALUE.ARRAY>
            <VALUE>Duplex Printing</VALUE>
            <VALUE>Black and White Printing</VALUE>
            $E:otherCapabilities
          </VALUE.ARRAY>
        </PROPERTY.ARRAY>
        <PROPERTY  NAME="HorizontalResolution">
          <VALUE>$S:horizontal</VALUE>
        </PROPERTY>
        <PROPERTY  NAME="VerticalResolution">
          <VALUE>$S:vertical</VALUE>
        </PROPERTY>
        <PROPERTY  NAME="MarkingTechnology">
          <VALUE>Laser</VALUE>
        </PROPERTY>
        <PROPERTY  NAME="PrintingSpeed">
          <VALUE>$S:speed</VALUE>
        </PROPERTY>
        $E:printerProperties
      </INSTANCE>,
   [$S:horizontal >= 1200],  [$S:vertical >= 1200],
   [$S:speed >= 25],
   notMember($S:errorState,  {"No Paper", "No Toner",
"Door Open", "Jammed",  "Service Requested" })
```

(b) XML serialisation.

Figure 8. XML clause C1: capability and status requirements of the role DocMaster.

6.1. Role requirement

Figure 8 gives an XML clause C_1, formalising capability and status requirements of DocMaster role. Both graphical and textual presentation of the clause is shown. However, for ease of understanding, only graphical presentation of XML clauses will be used in the sequel. Recall that a variable in an XML clause is preceded with '$', followed by its type and name. For example, **$S**:nodeX denotes a *String*-variable named nodeX and is instantiable into only a string, while **$E**:printerProperties is an *Expression*-variable instantiable into a list of XML expressions representing a sequence of objects or attributes. The given clause C_1 can be read as follows:

a) An actor playing DocMaster role can be installed into $S:nodeX, which is an instance of CIM_Printer,

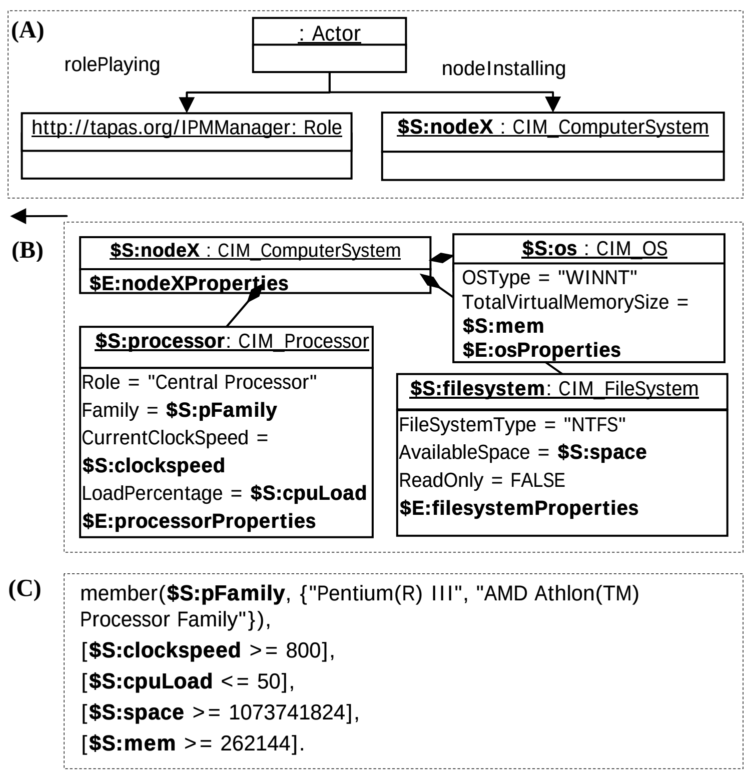
if

b) $S:nodeX is currently available and offers duplex printing and black-and-white printing capabilities, and laser marking technology,

c) the following additional conditions on $S:nodeX's capabilities and status are satisfied:

- [$S:horizontal >= 1200] and [$S:vertical >= 1200]: the horizontal and vertical resolutions of the print function are at least 1200 pixels per inch,
- [$S:speed >= 25]: the printing speed is greater than 25 pages per minute, and
- notMember($S:errorState, {"No Paper", "No Toner", "Door Open", "Jammed", "Service Requested"}): its current detected error state is not one of the given list.

The clauses C_2 of Figure 9 gives another simple example of modelling the capability and status requirement of the role IPMManager.



The clause specifies that:

(A) Any instance of the class CIM_ComputerSystem, represented by $S:nodeX, can install an actor for playing the role IPMManager, if the requirements (B) and (C) defined by the clause's body are met. That is:

- The central processor of that $S:nodeX is either Pentium III or AMD Athlon families with 800-MHz minimum clock speed and the load percentage less than 50.
- The installed operating system is WINNT with the minimum virtual memory of 262144 KB (256 MB).
- The computer's hosted file system must be NTFS with available space at least 1073741824 bytes (1 GB).

Figure 9. XML clause C2: capability and status requirements of the role IPMManager

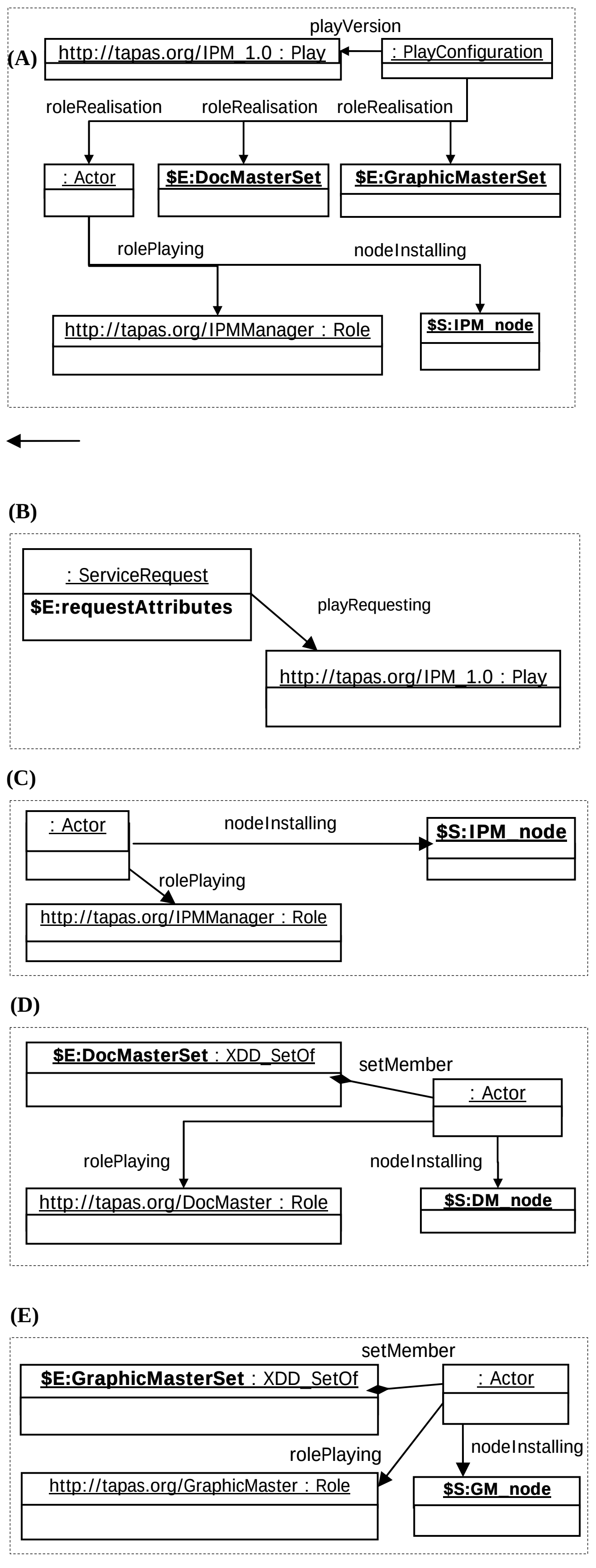
Note that due to space limitation and for the sake of simplicity, the paper illustrates only certain simple examples. For a more complete demonstrating example of configuration definitions, trouble reports as well as the computed (re)configuration plans, the reader is referred to the online prototype CM system available at http://tapas.item.ntnu.no/ipm.

6.2. Play configuration and reconfiguration rule

Figure 10 and Figure 11 present the play configuration and reconfiguration rules of the demonstrating IPM system, respectively.

6.3. Computing play configuration and reconfiguration plans

Let a play configuration definition for the IPM system be modeled as an XDD description which comprises the requirement specification of each role in a play as well as the play configuration constraints and the reconfiguration rule. Then, let the CSRep comprise CIM instances maintaining the capabilities/status of current, available nodes in a system. As an example of computing a configuration plan, assume that the CM receives the ServiceRequest of Figure 6 for installing and configuring the IPM system. A configuration plan computed by the CM by means of the prototype reasoning system is illustrated by Figure 12. It specifies that: (i) an actor playing the role IPMManager is to be installed at comp1, (ii) the role DocMaster at printerX and printerY, and (iii) the role GraphicMaster at printerX.



An XML clause defining a configuration rule of the play http://tapas.org/IPM_1.0.

The head (A) expresses that a valid configuration comprises the realisation of certain roles in the play, specified by three roleRealisation associations. The first association indicates that there must exist exactly one actor constituting the role IPMManager at a node $S:IPM_node. The other two associations, relating to the objects $E:DocMasterSet and $E:GraphicMaster, specify that the configuration also contains installation of actors realising some particular roles. The conditions on the number of actors to install and the roles to play are defined by the clause's body. The clause's body, comprising (B)–(E), specifies conditions for derivation of the defined configuration as well as its composition structure.

(B) indicates that the configuration will be computed upon the receipt of a ServiceRequest for installing such a version of the play.

(C) ensures that the node represented by $S:IPM_node has sufficient capabilities and status to install an actor for executing the role IPMManager. Obviously, this expression will be matched with the head of the clause C_2 of Figure 9, which models the IPMManager's requirement.

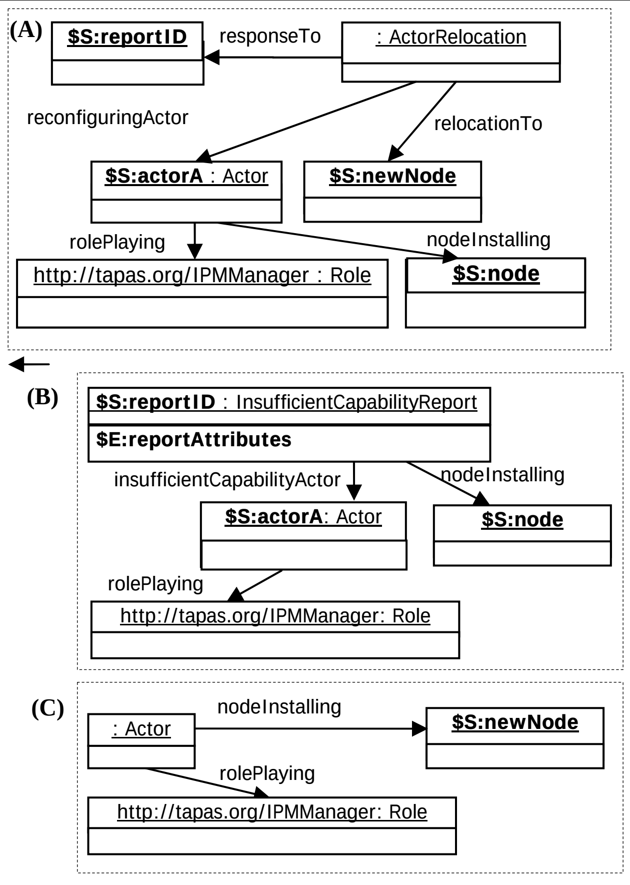
(D) specifies that $E:DocMasterSet represents a set of actors to be installed at nodes that are capable of playing the role DocMaster. That is, these nodes' capabilities and status must meet the requirement of the role represented by the clause C_1 of Figure 8. In this example, there can be more than one actor realising the role DocMaster.

Similarly, (E) specifies that $E:GraphicMasterSet represents a set of nodes capable of playing the role GraphicMaster. Note that a restriction on the number of roles that an instance can play is not defined. A particular printer may realise both DocMaster and GraphicMaster roles at the same time.

Figure 10. XML clause C3: a play configuration rule of the IPM service system.

7. Conclusions

A uniform representational and reasoning framework for dynamic configuration of service systems in TAPAS architecture has been developed, and its employment to model an Intelligent Printing Management system has been demonstrated. The framework enables services to be composed on the fly and the location for executing service components to be determined dynamically based on the offered capabilities as well as the current situation in the network. Moreover, during the service execution, it also permits adaptation of the service composition structure if certain significant events, such as a service component failure or QoS degradation, occur. In the framework, the Configuration Manager is the primary entity which reasons about the current system's capabilities & status, services' requirements and reconfiguration policies in order to dynamically generate appropriate service (re)configuration, hence enabling the system to cope with variations in the environment, achieve mandated performance levels and meet user satisfaction. To verify the framework's feasibility and potential in real applications, it has been implemented using the XET reasoning engine. Integration of the implemented framework into the *TAPAS platform* in order to provide a basis for experiments with dynamic configuration management is in progress.



The clause models a specific reconfiguration rule for handling InsufficientCapabilityReport of an actor playing the role IPMManager.It defines that:

(A) an ActorRelocation plan, specifying that the actor $S:actorA is to be relocated to $S:newNode, will be derive,

if

(B) there arises an InsufficientCapabilityReport, identified by $S:reportID and describing that the actor $S:actorA, currently playing the role IPMManager at the node $S:node, has insufficient capabilities to execute its functionality, and

(C) there exists a node in the system which is currently available and capable of playing the role IPMManager, and denote such a node by $S:newNode.

Figure 11. XML clause C4: a dynamic reconfiguration rule of the IPM service system.

(a) The generated XML document representing the computed plan.

```
<PlayConfiguration>
  <playVersion rdf:resource="http://tapas.org/IPM_1.0"/>
    <roleRealisation>
      <Actor>
        <rolePlaying rdf:resource="http://tapas.org/IPMManager"/>
        <nodeInstalling rdf:resource="http://comp1.tapas.org"/>
      </Actor>
    </roleRealisation>
    <roleRealisation>
      <Actor>
        <rolePlaying rdf:resource="http://tapas.org/DocMaster"/>
        <nodeInstalling rdf:resource="http://PrinterY.tapas.org"/>
       <nodeInstalling rdf:resource="http://PrinterX.tapas.org"/>
      </Actor>
    </roleRealisation>
    <roleRealisation>
      <Actor>
        <rolePlaying
               rdf:resource="http://tapas.org/GraphicMaster"/>
        <nodeInstalling
               rdf:resource="http://PrinterX.tapas.org"/>
      </Actor>
     </roleRealisation>
</PlayConfiguration>
```

(b) Corresponding RDF graph.

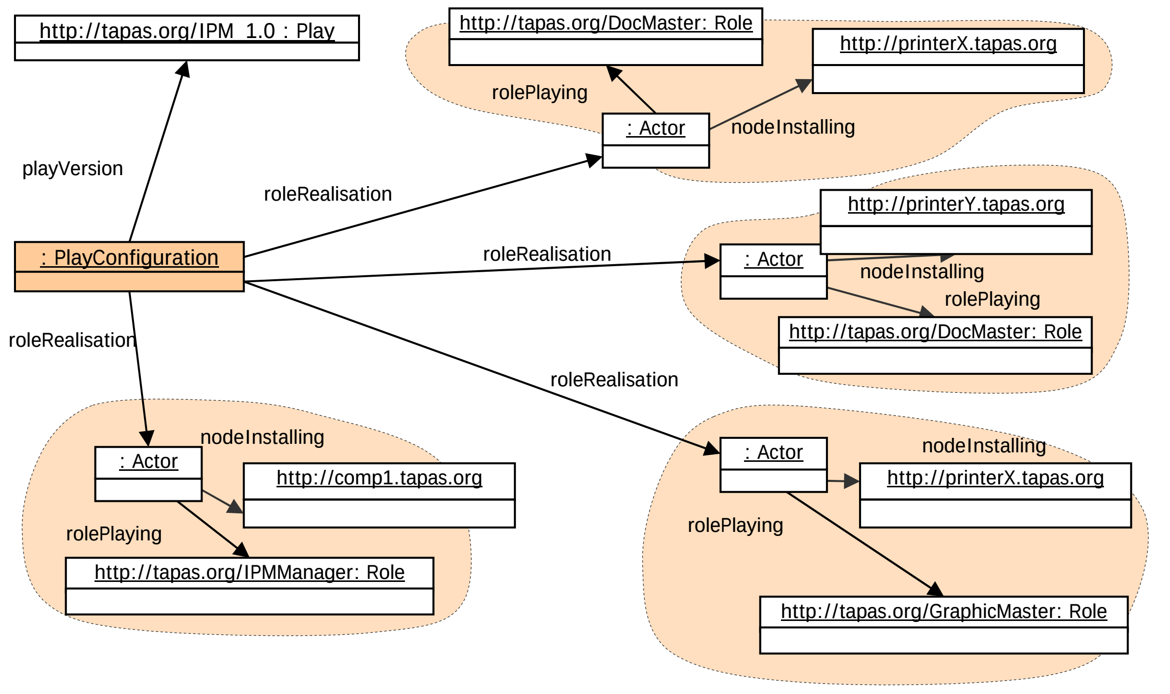


Figure 12. XML clause C4: a dynamic reconfiguration rule of the IPM service system.

References

Aagesen, F. A., C. Anutariya, et al. (2002). Support Specification and Selection in TAPAS. IFIP WG6.7 Workshop and Eunice Summer School on Adaptable Networks and Teleservices, Trondheim, Norway, Tapir.

Aagesen, F. A., B. E. Helvik, et al. (2003). On Adaptable Networking. Int'l Conf. on Information and Communication Technologies (ICT 2003), Assumption University, Thailand.

Aagesen, F. A., B. E. Helvik, et al. (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.

Aagesen, F. A., B. E. Helvik, et al. (1999). Towards a Plug and Play Architecture for Telecommunications. 5th IFIP Conf. Intelligence in Networks (SmartNet 99), Bangkok, Thailand, Kluwer Academic Publisher.

Anutariya, C., V. Wuwongse, et al. (2002). An Equivalent-Transformation-Based XML Rule Language. Int’l Workshop Rule Markup Languages for Business Rules in the Semantic Web, Sardinia, Italy.

Bakour, H. and N. Boukhatem (2004). ASMA: An Active Architecture for Dynamic Service Deployment. IFIP Int’l Conf. Intelligence in Communication Systems (INTELLCOMM 2004), Bangkok, Thailand.

Berners-Lee, T., M. Fischetti, et al. (1999). Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor, Harper, CA.

Bieszczad, A., B. Pagurek, et al. (1998). "Mobile Agents for Network Management." IEEE Communications Surveys **1**(1).

Brickley, D. and R. V. Guha (2004). RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation 10 February 2004. B. McBride.

Cohen, R. and D. Raz (2004). An Open and Modular Approach for a Context Distribution System. Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), Seoul, Korea.

D’Antonio, S., M. D’Arienzo, et al. (2004). An Architecture for Automatic Configuration of Integrated Networks. Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), Seoul, Korea.

Hendler, J. and D. McGuinness (2000). "The DARPA Agent Markup Language." IEEE Intelligent Systems **15**(2): 72-73.

Inoue, Y., M. Lapierre, et al. (1999). The TINA Book: A Co-operative Solution for a Competitive World, Prentice Hall.

Keller, A., J. L. Hellerstein, et al. (2004). The CHAMPS System: Change Management in Planning and Scheduling. Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), Seoul, Korea.

Lassila, O. and R. R. Swick (1999). Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999.

McGuinness, D. L. and F. Harmelen (2004). OWL Web Ontology Language Overview, W3C Recommendation 10 February 2004.

Raza, S. K. and A. Bieszczad (2004). Network Configuration with Plug and Play Components. Proc. 6th IFIP/IEEE Network Operations and Management Symposium (NOMS 2004), Seoul, Korea.

Sahai, A., S. Singhal, et al. (2004). Automated Policy-Based Resource Construction in Utility Computing Environments. IEEE/IFIP Network Operations and Management Symposium NOMS'2004, Seoul, South Korea.

Shiaa, M. M. (2004). Mobility Support Framework in Adaptable Service Architecture. IEEE/IFIP Net-Con' 2003, Muscat, Oman.

Shiaa, M. M. and F. A. Aagesen (2002). Mobility Management in Plug and Play Network Architecture. Proc. IFIP 7th Int'l Conf. Intelligence in Networks (SmartNet 2002), Saariselka, Finland, Kluwer Academic Publishers.

Shiaa, M. M., S. Jiang, et al. (2004). An XML-based Framework for Dynamic Service Management. The 2004 IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM 04), Bangkok, Thailand.

Solarski, M., L. Strick, et al. (2004). Flexible Middleware Support for Future Mobile Services and Their Context-Aware Adaptation. IFIP Int'l Conf. Intelligence in Communication Systems (INTELLCOMM 2004), Bangkok, Thailand.

Tennenhouse, D. L., J. M. Smith, et al. (1997). "A Survey of Active Network Research." IEEE Communications **35**(1).

Westerinen, A. and J. Strassner (2000). "Common Information Model (CIM) Core Model Distributed Management Task Force White Paper Version 2.4."

Wuwongse, V., K. Akama, et al. (2003). "A Data Model for XML Databases." Intelligent Information Systems **20**(1): 63-80.

Wuwongse, V., C. Anutariya, et al. (2001). "XML Declarative Description (XDD): A Language for the Semantic Web." IEEE Intelligent Systems **16**(3): 54-65.

Paper C

A Framework for Dynamic Service Composition

Paramai Supadulchai and Finn Arve Aagesen

In Proceedings of the first international IEEE WoWMoM Workshop on *Autonomic Communications and Computing* (ACC 2005) in conjunction with IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks WoWMoM 2005

Taormina, Italy, June 13 2005

A Framework for Dynamic Service Composition

Paramai Supadulchai and Finn Arve Aagesen
Department of Telematics, Norwegian University of Science and Technology (NTNU)
{paramai, finnarve}@item.ntnu.no

Abstract

To be able to utilize the generative potential of future networks for service composition, the attributes of services and networks must be appropriately formalized, stored and made available. Important attributes are the capability and the status. A capability is an inherent property of a node or a user, which defines the ability to do something. A capability in a network node is a feature available to implement services. A capability of a user is a feature that makes the user capable of using services. Status is a measure for the situation in a system. This paper proposes a representation framework for capability and status, denoted as Unified Capability and Status Representation Framework (UniCS). This framework is used to decide upon dynamic use of capabilities, and is used to support the dynamic composition of a service system. UniCS consists of facts and configuration rules. The facts describe the availability and requirement of capabilities and status of a service system. The configuration rules verify, manipulate, transform and discover new facts with defined axioms and constraints. An instance of UniCS is the input specification for a reasoning engine to dynamically generate a composition plan for a service system.

1. Introduction

A network-based service system consisting of services, service components and nodes is considered. A *service* is realized by the structural and behavioral arrangement of *service components*, which by their inter working provide a service in the role of a *service provider* to a *service user*. Service components are executed as *software components* in *nodes*, which are physical processing units such as servers, routers, switches and user terminals. User terminals can be phones, laptops, PCs and PDAs etc.

Traditionally, the nodes as well as the service components have a predefined functionality. However, changes are taking place. Nodes are getting more generic and can have any kind of capabilities such as MP3, camera and storage. The software components have been also changed from being *static* components to become more *dynamic* and be able to download and execute different functionality depending on the need. Such generic programs are from now on denoted as *actors*. The name actor is chosen because of the analogy with the actor in the theatre, which is able to play different roles in different plays.

We are entering a generative era, which gives a high degree of flexibility. To utilize the generative potential, the attributes of services, service components, software components and nodes must be appropriately formalized, stored and made available. As a first further step towards this formalization, the concepts capability and status are introduced.

A capability is an inherent property of a node or a user, which defines the ability to do something. A capability in a node is a feature available to implement services. An actor executes a program, which may need capabilities in the node. A capability of a user is the feature that makes the user capable of using services. Capabilities can be classified into:

- Resources: physical hardware components with finite capacity,
- Functions: pure software or combined software/ hardware component performing particular tasks,
- Data: just data, the interpretation, validity and life span of which depend on the context of the usage.

Status is a measure for the situation in a system with respect to the number of active entities, the traffic situation and the Quality of Services (QoS) etc. Status reflects an instantaneous state of the system.

Rather than using the traditional approach that nodes and service components have a pre-defined functionality, the functionality can be composed from several cooperating actors hosted in nodes. We propose a representation framework for capability and status, denoted as Unified Capability and Status Representation Framework (UniCS). This framework is used to decide upon dynamic use of capabilities, and is used to support the dynamic composition of a service system. UniCS consists of facts and configuration rules. The facts describe the availability and requirement of capabilities and status of a service system. The configuration rules verify, manipulate, transform and discover new facts with defined axioms and constraints. An instance of UniCS is the specification given as an input to a reasoning engine to generate a composition plan for a service system.

The work presented in this paper has been related to the Telematics Architecture for Play-based Adaptable System (TAPAS) [1]. Section 2 discusses related work. Section 3 gives some TAPAS concepts, which are extensions to the generic concepts already defined. Section 4 gives an overview of UniCS. Section 5 describes the methodology of the dynamic composition of a service system. Section 6 gives a summary and presents our conclusions.

2. Related work

Several research activities are related to capability representations [5,8,9,11]. A similar work to UniCS presented in this paper in term of objectives, functionalities and architecture, is a Resource Definition Framework (RDF)-based knowledge model for network management [5], which provides an analogous framework to describe capability facts in RDF. However, it requires additional framework(s) to describe configuration rules. Directory Enabled Network NGOSS (DEN-ng) [9] describes the configuration rule partially in term of constraints in a specific language, i.e. Object Constraint Language (OCL). However, OCL limits the power of DEN-ng to only verifying facts, while not permitting it to manipulate, transform or discover them.

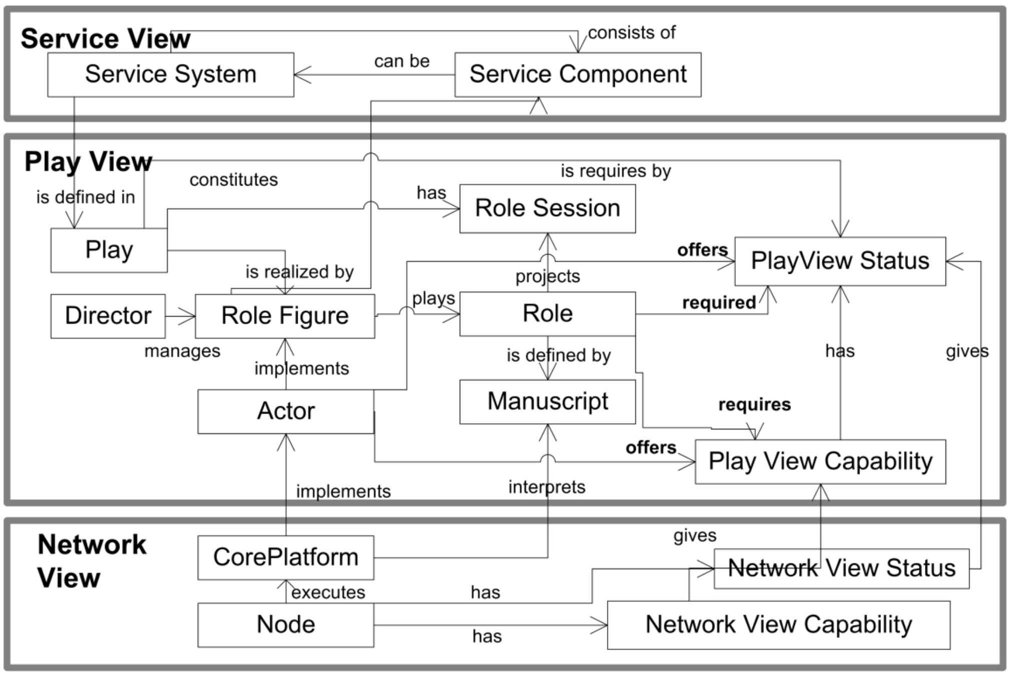


Figure 1 the TAPAS computing architecture

3. Necessary TAPAS concepts

The Telematics Architecture for Play-based Adaptable System (TAPAS) intends to be an architecture for *autonomic network-based systems* that gives *rear-rangement flexibility*, *failure robustness* and *resource load awareness and control* [1]. In analogy with the TINA architecture [3], the TAPAS architecture is separated into a *service architecture* and a *computing architecture* as follows.

- The *service* architecture is an architecture showing the structure of *services* and *services components*.
- The *computing* architecture is a generic architecture for the modeling of any service *software components*.

These architectures are not independent and can be seen as architectures at different abstraction layers. The service architecture, however, has focus on the functionality independent of implementation, and the computing architecture has focus on the modeling of functionality with respect to implementation, but independent of the nature of the functionality.

The relationship of services and service components in the service architecture are realized by the computing architecture, which will be the focus of this paper.

3.1. TAPAS computing architecture

TAPAS computing architecture has three layers: the *service view*, the *play view* and the *network view* as shown in Figure 1. **The service view** concepts are rather generic and should be consistent with any service architecture. **The network view** concepts are consistent with any corresponding network architecture, with exception of the *core platform*, which is a specific platform supporting the play view concepts. The network view consists of *nodes*, which are typically network processing units such as mobile phone, desktop computer, laptop, printer and router that possess particular *Network View Capabilities*, from now on abbreviated as *NV-Capabilities*. Nodes are installed in the *core platform.* At a specific time point, a

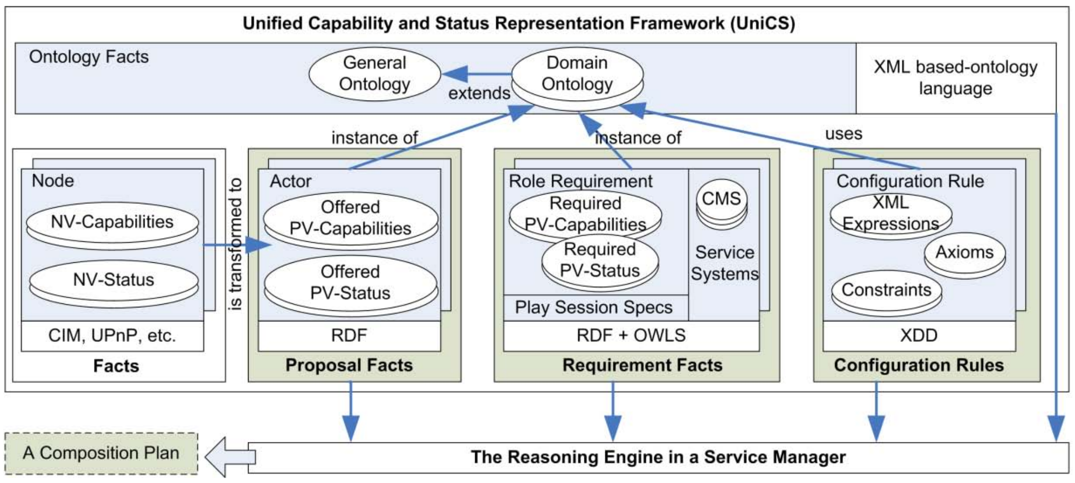


Figure 2 the UniCS framework

Network View Status denoted as *NV-Status* is the state of a system with respect to the number of active entities, traffic situation and QoS etc.

The play view is a basis for designing functionality that can meet the *rearrangement*, the *robustness*, the *survivability*, the *QoS* awareness and *resource control* properties. The play view concepts are seemingly rearrangement flexibility oriented. The *capability and status* concepts, however, also give a basis for the further design of the robustness, the survivability, the QoS awareness and resource control properties.

3.2. TAPAS theater metaphor

The play view is founded on a theater metaphor. The TAPAS actor is a generic software component consistent with the actor definition given in Section 1. However, the TAPAS actor is specialized as follows. Actors perform *roles* according to predefined manuscripts, and a director manages their performance. Actors are software components in the nodes that can download manuscripts. They have *Play View Capabilities and Status* abbreviated as *PV-Capabilities and -Status*, which are transformed from NV-Capability and -Status of the nodes. The transformation is also based on UniCS and is referred to [10]. An actor will constitute a role figure by behaving according to a manuscript that defines the functional behavior of that particular role in a play. A role session is a projection of the behavior of a role figure with respect to one of its interacting role figures. Actors can be moved between nodes and their role sessions can be re-instantiated automatically [6].

A director is an actor with supervisory status regarding other actors. When the director needs to choose a fitting actor for a certain role figure, he requests help from *a service manager*, which is a dedicated role figure to generate a composition plan for the dynamic service composition. The director reads the generated composition plan and assigns role-based manuscripts to the recommended actors. The actor interprets the manuscript and behaves accordingly. The utilization of manuscripts is beyond the scope of this paper and is referred to [7].

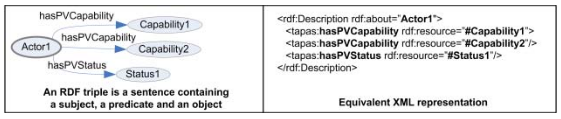
A service system is defined by *a play*. A play consists of several actors playing different roles, each possibly having different PV-Capabilities and –Status requirements. *An actor will constitute a role figure, which will constitute a service component based on a role defined by a* *manuscript*. The ability of an actor to play a role depends on the matching of the required PV-Capabilities and -Status of the role and the offered PV-Capabilities and -Status of the actor [1].

4. Unified Capability and Status Representation Framework (UniCS)

As already defined in Section 3.2, the behaviors of service components are based on *roles*. To allocate an actor to a specific role, the information of available actors and their PV-Capabilities and -Status is needed. This capability and status information must be described in a *formal* and *machine-understandable* way. *Unified Capability and Status Representation Framework (UniCS)* as shown in Figure 2 is a *unified* representation and an *executable* framework for capability and status information.

UniCS is used to represent the requirement on how to dynamically compose a service system. This requirement is given to the reasoning engine in a service manager to generate a composition plan, which suggests appropriate actors to play roles and become the service components of the service system.

4.1 Syntactic representation


hasPVCapability
Capability1
hasPVCapability
Actor1
Capability2
hasPVStatus
Status1
An RDF triple is a sentence containing a subject, a predicate and an object
Equivalent XML representation

```
<rdf:Description rdf:about="Actor1">
  <tapas:hasPVCapability rdf:resource="#Capability1">
  <tapas:hasPVCapability rdf:resource="#Capability2"/>
  <tapas:hasPVStatus rdf:resource="#Status1"/>
</rdf:Description>
```

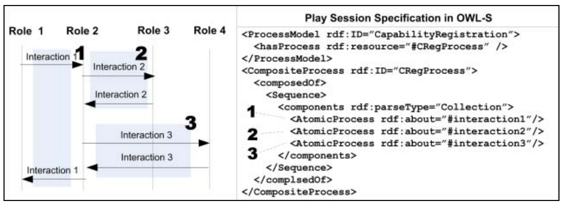
Figure 3 the PV-Capability and -Status representation

UniCS consists of *facts* and *configuration rules* providing a way to separate syntactic and semantic representation respectively. Facts indicate relationships in a system. An example of a fact is *"printer A" "has capability" "duplex printing"*. Concerning only capability and status, facts can be represented in various XML syntaxes. In the network view, any syntax chosen by the manufacturers or the network management program can be used. The examples are Common Information Model encoded in XML (CIM-XML) [11] and Universal Plug-and-Play (UPnP) [8].

In the play view, PV-Capability and -Status are the projection of NV-Capability and -Status. Facts concerning actors and PV-Capabilities and -Status are from now on classified as *proposal facts*. The PV-Capability syntax is carried on RDF. The main reason is because representing facts transformed from various NV-Capabilities and –Status syntax is rather easy with the RDF construct triple [10]. Facts in RDF can also be seamlessly facilitated with *additional domain ontology* from any XML-based ontology language.

The facts concerning roles in a service system are defined as *play session specification* and *role requirement*. A play session specification is a set of all *role sessions* in a play constituting a service component. A play session is specified by *an atomic process* in the *Web Ontology Language for Semantic Web (OWL-S)* [4] and has two associative roles: *invoker* and *server*. The invoker role generally has no PV-Capability and -Status requirement. The server role requires specific PV-Capabilities and -Status. Figure 4 shows a play session specification example.

At the time of writing, OWL-S is incapable of describing the server roles' required PV-Capabilities and -Status, which are essential criteria in the composition of a service system. We propose the using role requirement, modeled in RDF, as an extension to each play session to describe the roles' required PV-Capabilities and –Status. Role requirement representation is similar to Figure 3 except that an actor is replaced by a role as the subject of the fact. Facts concerning the play session specification and the role requirement of a service system are classified as *requirement facts*. The requirement facts and the proposal facts will be matched by *configuration rules*.



```
<ProcessModel rdf:ID="CapabilityRegistration">
  <hasProcess rdf:resource="#CRegProcess" />
</ProcessModel>
<CompositeProcess rdf:ID="CRegProcess">
  <composedOf>
    <Sequence>
      <components rdf:parseType="Collection">
        <AtomicProcess rdf:about="#interaction1"/>
        <AtomicProcess rdf:about="#interaction2"/>
        <AtomicProcess rdf:about="#interaction3"/>
      </components>
    </Sequence>
  </complsedOf>
</CompositeProcess>
```

Figure 4 play session specification

4.2. Semantic representation

Table 1 Types of XML variables

Type	Instantiation and examples
N	*XML element or attribute names* Ex: `<$N:var1>…</$N:var1>` can be instantiated to `<actor>...</actor>` or `<node>...</node>`
S	*XML string* Ex: `<prop name="$S:var1"/>` can be instantiated into `<prop name="prop1"/>` or `<prop name="prop2"/>`
P	*Sequence of zero or more attribute-value pairs* Ex: `<element $P:var1/>` can be instantiated into `<element/>` or `<element name="1"/>`
E	*Sequence of zero or more XML expressions* Ex: `<element>$E:var1</element>` can be instantiated into `<element/>` or `<element><value>1</value></element>`
I	*Part of XML expressions* Ex: `<$I:var1><attr/></$I:var1>` can be instantiated into `<element><prop><attr/></prop></element>`

Configuration rules are defined in a semantic web language, XML Declarative Description (XDD) [12]. XDD is an XML-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 expression without variables – are called ground XML expressions. Every component of an XML expression can contain variables as in Table 1. Every variable is prefixed with '$*T*:' where *T* denotes its type.

A configuration rule is an XML clause of the form:

$$H, \{C_1, \ldots C_m\} \rightarrow B_1, \ldots B_n$$

where $m, n \geq 0$, H and B_i are XML expressions. And each of the C_i is a predefined XML condition used to limit the rule for a certain circumstances. This allows the modeling of *constraints* for a rule. *Axioms* are defined from one or more rule(s) [11]. The XML expression H is called the *head* of the clause. The set of B_i is the body of the clause. When the body is empty, such a clause is referred to an XML unit clause, and the symbol '$\rightarrow$' will be omitted. Hence any facts in form of XML elements or documents can be mapped directly onto a ground XML unit clause.

4.3 UniCS reasoning mechanism

Intuitively, the UniCS reasoning process begins with an XML expression based query. The reasoning engine formulates an XML clause from the query of the form:

$$Q \leftarrow Q$$

The XML expression Q represents the constructer of the expected answer which can be derived *if* all the bodies of the clause hold. However, if one or more XML expression bodies still contain XML variables. These variables must be matched and resolved from other rules.

A body from the query clause will be matched with the head of each rule. At the beginning, there is only one body *Q*. Consider a rule R_1 in the form:

$$R_1: H, C_1 \rightarrow B_1, B_2$$

If the XML structure of the body *Q* of the clause and the head *H* of the rule R_1 match without violating condition C_1, the body *Q* will be transformed into B_1 and B_2. All XML variables in the head *Q* and the new bodies B_1 and B_2 of the query clause will be instantiated. The query clause will be in the form:

$$Q^* \leftarrow B_1^*, B_2^*$$

Where X^* means the one or more variables in the XML expression *X* has been instantiated and removed.

The transformation process ends when either 1) the query clause has been transformed into a unit clause or 2) there is no rule that can transform the current bodies B_i of the query clause. If the constructor *Q* is transformed successfully into Q_f that contain no XML variable, the reasoning process ends and a desired answer is obtained. Due to the space limitation, the details how the reasoning engine performs the rule matching and the variable instantiation will not be presented in this paper. The reader should be referred to [12].

5. Service composition

Three generic configuration rules for composing any service system are formulated. For each rule, a graphical representation of RDF triples together with the various types of XML variables is used instead of the equivalent XML clause due to the space limitation. These rules and the query clause are as follows:

5.1. The query clause (Figure 5):

Figure 5 the graphical representation of the query clause

The query clause contains the name of the service system to be composed, which can be changed. The meaning of both head and body of the clause is that "*Service System 1*" can have a role "*$S:Role*", which can be played by an actor "*$S:Actor*".

5.2. Rule 1 (Figure 6):

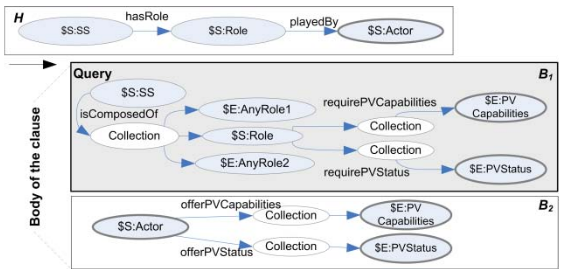


Figure 6 the graphical representation of rule 1

The head *H* of Rule 1 is similar to the body B_1 of the query clause except the variable *$S:SS* that makes this rule applicable to any service system. After the variable *$:SS* has been instantiated with the name of a service system from the query clause, the body B_1 will query the *requirement facts* and find the play session specification and the role requirement of that service system. Each role in *$S:SS* requires PV-Capabilities and -Status, represented by *$E:PV-Capabilities* and *$E:PV-Status*. The body B_2 looks for the *proposal facts* with the capabilities and status represented by the same E-variables. Rule 1 matches the proposal and requirement facts that refer to the same variables.

5.2. Rule 2 (Figure 7):

Rule 2 is for querying the actor that has a set of PV-Capability and -Status, represented by *$E:PV-Capability* and *$E:PV-Status*. The body B_1 will query all the proposal facts and find actors that have such a qualification. Additional E-variables in B_1 allows an actor to have PV-Capabilities and -Status in addition to those in *$E:PV-Capabilities* and *$E:PV-Status*.

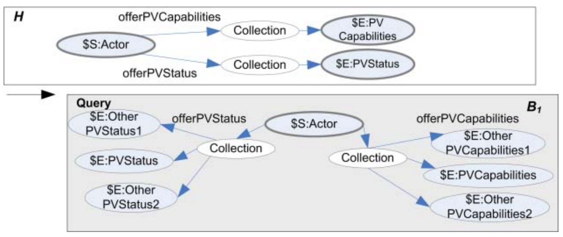


Figure 7 the graphical representation of rule 2

5.4 The result (Figure 8)

After the execution, the composition plan of *"Service System 1"* is produced as illustrated in Figure 8. Note that Role 2 can be played by both *Actor B* and *C* because they both have the required PV-Capabilities and -Status.

We used XML Equivalent Transformation (XET) [2, 12], a Java-based reasoning engine that transforms the query clause by the XDD-based rules to compose a plan for Capability Management System, a service system that manages capabilities in TAPAS.

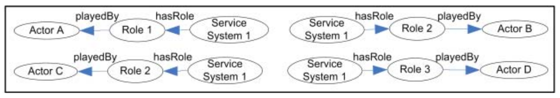


Figure 8 the graphical representation of the composition plan

6. Conclusion

This paper has presented a framework for capability and status representation, denoted as *Unified Capability and Status Representation Framework* (UniCS). This framework has been further used to support the modeling of the dynamic composition of service systems. UniCS consists of facts and configuration rules. Facts are categorized *as proposal facts*, which are actors, PV-Capabilities and -Status, and *requirement facts*, which are play session specification and role requirement. Configuration rules map the proposal facts and the requirement facts and discover a *composition plan*. As a result, a service system can be *dynamically* composed.

References

[1] Aagesen, F.A., et al., On Adaptable Networking. ICT'2003, Assumption University, Thailand, 4/2003.

[2] Anutariya, C., et al., An Equivalent-Transformation-Based XML Rule Language. Int'l Workshop Rule Markup Languages for Business Rules in the Semantic Web, Sardinia, Italy, 6/2002.

[3] Inoue, Y., et al., The TINA Book. A Co-operative Solution for a Competitive World. *Prentice Hall*, 1999.

[4] OWL Service Coalition, Semantic Markup for Web Services, 11/2003.

[5] Shen, J. and Y. Yang, RDF-Based Knowledge Model for Network Management. IFIP/IEEE IM 2003. Colorado, Springs, 3/2003.

[6] Shiaa, M.M., Mobility Support Framework in Adaptable Service Architecture. IEEE/IFIP Net-Con'2003, Muscat, Oman, 10/2003.

[7] Shiaa, M.M., et al., An XML-based Framework for Dynamic Service Management. IFIP INTELLCOMM 2004, Bangkok, Thailand, 11/2004.

[8] Steinfeld, E.F., Devices that play together, work together, *EDN Magazine*, 9/2001.

[9] Strassner, J., DEN-ng: Achieving Business-Driven Network Management. IEEE/IFIP NOMS 2002, Florence, Italy, 4/2002.

[10] Supadulchai, P., Aagesen, F.A., An Approach to Capability and Status Modeling, NIK 2004, Stavanger, Norway, 11/2004.

[11] Westerinen, A. and W. Bumpus, The Continuing Evolution of Distributed Systems Management. *IEICE TRANS. INF & SYST.*, Vol. E86-D Nr. 11: 11/2003.

[12] Wuwongse, V., et al.: XML Declarative Description (XDD), A Language for the Semantic Web. *IEEE Intelligent Systems*, Vol. 16 Nr.3, 5-6/2001.

Paper D

Autonomic Service Configuration by a Combined State Machine and Reasoning Engine-based Actor

Paramai Supadulchai and Finn Arve Aagesen

In Proceedings of *the 2005 IFIP International Conference on Intelligence in Communication Systems* (INTELLCOMM 2005)

Delta Centre-Ville Hotel Montréal, Canada, October 17-19, 2005

Autonomic Service Configuration by a Combined State Machine and Reasoning Engine Based Actor

Paramai Supadulchai and Finn Arve Aagesen

NTNU, Department of Telematics, N-7491 Trondheim, Norway

Abstract: Service systems constituted by service components are considered. Service components are executed as software components in nodes, which are physical processing units such as servers, routers, switches and user terminals. A capability is an inherent property of a node or a user, which defines the ability to do something. Status is a measure for the situation in a system. A service system has defined requirements to capabilities and status. Because of continuous changes in capabilities and status, dynamic service configuration with respect to capabilities and status is needed. Software components are generic components, denoted as actors. An actor is able to download, execute and move functionality, denoted as a role. Configuration is based on the matching between required capability and status of a role and the present executing capabilities and status of nodes. We propose an approach for role specification and execution based on a combination an Extended Finite State Machine and a rule based reasoning engine. Actor execution support consisting of a state machine interpreter and a reasoning engine has been implemented, and has also been applied for a service configuration example.

1. **Introduction**

Service systems constituted by *service components* are considered. Service components are executed as software components in *nodes,* which are physical processing units such as servers, routers, switches and user terminals such as phones, laptops, PCs, and PDAs. Traditionally, the nodes as well as the service components have a predefined functionality. However, changes are taking place. Nodes are getting more generic and can have any kind of capabilities such as MP3, camera and storage. The software components have been also changed from being static components to become more dynamic and be able to download and execute different functionality depending on the need. Such generic programs are from now on denoted as *actors*. The name actor is chosen because of the analogy with the actor in the theatre, which is able to play different *roles* play defined in different *plays*.

To utilize the flexibility potential, the attributes of services, service components, software components and nodes must be appropriately formalized, stored and made available. As a first further step towards this formalization, the concepts *status and capability* are introduced.

Status is a measure for the situation in a system with respect to the number of active entities, the traffic situation and the Quality of Service. A *capability* is an inherent property of a node or a user, which defines the ability to do something. A capability in a node is a feature available to implement services. A capability of a user is a property that makes the user capable of using services. An actor executes a program, which may need capabilities in the node. Capabilities can be classified into:

- *Resources:* physical hardware components with finite capacity,
- *Functions:* pure software or combined software/hardware component performing particular tasks,
- *Data:* just data, the interpretation, validity and life span of which depend on the context of the usage.

The functionality to be played by an actor participating in the constitution of a service is denoted as its role. We use the role-figure as a generic concept for the actor which is playing a role. So services and service components are realized by role-figures. *Service configuration* is here the configuration of services with respect to the required capability and status of the roles.

The Role of an actor is defined in a manuscript, which consists of an EFSM *(Extended Finite State Machine)* extended with rule-based policies. Using a local *rule-based reasoning* engine adds the ability to cope with various situations more flexible than is possible by the pure EFSM. Actors can locally take place in the configuration and reconfiguration of the services, in which they are a part of. The reasoning engine is based an XET (XML Equivalent Transformation) rule-based language.

The work presented in this paper has been related to the Telematics architecture for Play-based Adaptable System (TAPAS) [2]. Section 2 discusses related work. Section 3 presents the model used for the combined EFSM and reasoning engine based actor. Section 4 gives a short presentation of the TAPAS architecture with focus on the elements relevant for the autonomic service configuration. Section 5 presents the data model. Section 6 presents a simple scenario where an actor actively participates in service reconfiguration. Section 7 gives a summary and presents our conclusions.

2. Related work

The mobility of service components have been dealt within a number of approaches. An example is the Intelligent Agent, which is the most related to our work. DOSE [4] is an agent-based autonomic platform that uses Semantic Web to come up with response to failures. However, the behavior of each service component must be fixed along with the moving codes that cannot be downloaded or changed. A technique to overcome this shortcoming has been proposed using Java Reflection [5]. The behavior of server components can be downloaded or changed based up on a reasoning mechanism. However, the reasoning mechanism itself cannot be downloaded or altered. In our approach, the behavior of service components and the rules used in the reasoning mechanism are downloadable and can be changed upon needs.

3. The actor model

The actor role is defined as an Extended Finite State Machine (EFSM) extended with policies. The mechanism interpreting the manuscript is an EFSM interpreter extended with a reasoning mechanism. The data structure applied for the representation of an EFSM is shown in Figure 1. An EFSM contains the EFSM name, 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 actor will perform a number of actions, and/or send a number of outputs, and then go to the next state. Actions are functions and tasks performed during a specific state: computation on local data, role session initialization, message passing, etc. The structure of the `<ACTIONS>` list specifies the name, the parameters and the classification of an action.

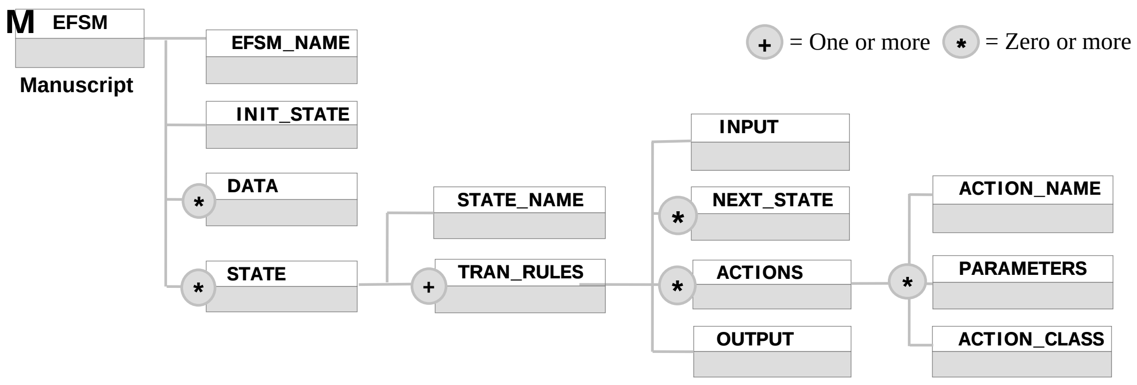


Figure 1. Data structure of EFSM-based manuscript

Rule-based reasoning is considered as a special type of EFSM action that executes policies. Policies are expresses in the XML Equivalent Transformation language (XET) [3]. The reasoning engine can directly operate and reason about XET descriptions.

The XET language is an XML-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 expression without variables, are denoted as ground XML expressions. Every component of an XML expression can contain variables as shown in Table 1. Every variable is prefixed with '*T*var_' where *T* denotes its type.

Table 1. Types of XML variables

Type	Instantiation and examples
N	XML element or attribute names Ex: `<Nvar_X>…</Nvar_X>` can be instan-

Type	Instantiation and examples
	tiated to `<div>…</div>` or `<span>…</span>`
S	XML string Ex: `<a name='Svar_Y'/>` can be instantiated into `<a name='http:// …'/>` or `<a name='ftp:// …'/>`
P	Sequence of zero or more attribute-value pairs Ex: `<p Pvar_Z='NULL'/>` can be instantiated into `<p/>` or `<p style='…'/>`
E	Sequence of zero or more XML expressions Ex: `<p>Evar_P</p>` can be instantiated into `<p/>` or `<p><div>…</div><br/><br/></p>`
I	Part of XML expressions Ex: `<Ivar_X><hr/></IvarX>` can be instantiated into `<body><hr/></body>` or `<hr/>`

A rule is an XML clause of the form:

$$H, \{C_1, \ldots C_m\} \rightarrow B_1, \ldots B_n$$

where m, $n \geq 0$, H and B_i are XML expressions. And each of the C_i is a predefined XML condition used to limit the rule for a certain circumstances. This allows constraints modeling for a rule. Axioms are defined from one or more rule(s). The XML expression H is called the head of the clause. The B_i is a body atom of the clause. When the list of body atom is empty, such a clause is referred to an XML unit clause, and the symbol '$\rightarrow$' will be omitted. Hence ordinary XML elements or documents can be mapped directly onto a ground XML unit clause.

The reasoning process begins with an XML expression-based query. An XML clause will be formulated from the query in form:

$$Q \rightarrow Q$$

XML expression Q represents the constructer of the expected answer which can be derived if all the body atoms of the clause hold. However, if one or more XML expression body atoms still contain XML variables. These variables must be matched and resolved from other rules.

A body from the query clause will be matched with the head of each rule. At the beginning, there is only one body Q. Consider a rule R_1 in the form:

$$R_1: H, \{C_1\} \rightarrow B_1, B_2$$

If the XML structure of the body Q of the clause and the head H of the rule R_1 match without violating condition C_1, the body Q will be transformed into B_1 and B_2. All XML variables in the head Q and the new bodies B_1 and B_2 of the query clause will be instantiated. The query clause will be in the form:

$$Q^* \rightarrow B_1^*, B_2^*$$

Where X^* means the one or more variables in the XML expression X has been instantiated and removed.

The transformation process ends when either 1) the query clause has been transformed into a unit clause or 2) there is no rule that can transform the current bodies B_i of the query clause. If the constructor Q is transformed successfully into Q_f that contain no XML variable, the reasoning process ends and a desired answer is obtained.

4. TAPAS architecture

"Adaptable service systems" are service systems that adapts dynamic to changes in both time and position related to Users, Nodes, Capabilities, Status and Changed Service Requirements. Adaptability can be modeled as a property consisting of 3 property classes: 1) rearrangement flexibility, 2) failure robustness and survivability, and 3) QoS awareness and resource control. The Telematics Architecture for Play-based Adaptable System (TAPAS) intends to meet these properties [2]. In analogy with the TINA architecture [6], the TAPAS architecture is separated into a system management architecture and a computing architecture as follows:

- The system management architecture is an architecture showing the structure of services and services components.
- The computing architecture is a generic architecture for the modeling of any service software components.

These architectures are not independent and can be seen as architectures at different abstraction layers. The system management architecture, however, has focus on the functionality independent of implementation, and the computing architecture has focus on the modeling of functionality with respect to implementation, but independent of the nature of the functionality.

4.1 Computing architecture

TAPAS computing architecture has three layers: the service view, the play view and the network view as illustrated in Figure 2. For details see [1].

A service system consists of service components and the network system consists of nodes. *The play view* is the intended basis for designing functionality that can meet the adaptability properties as defined above. The play view is founded on the theater metaphor introduced in Sec.1. TAPAS actors are software components in nodes that can download manuscripts. An actor that does not have a role assigned is denoted as a *free actor*. An actor playing a role in a manuscript is denoted as a *role figure*. A service system is constituted by a play, and leaf service component are constituted by role figures. A *role session* is the dialog between two executing role figures. A role figure can move between nodes and its role sessions can be re-instantiated automatically. This mechanism, however, is not the focus of this paper. It is referred to [7].

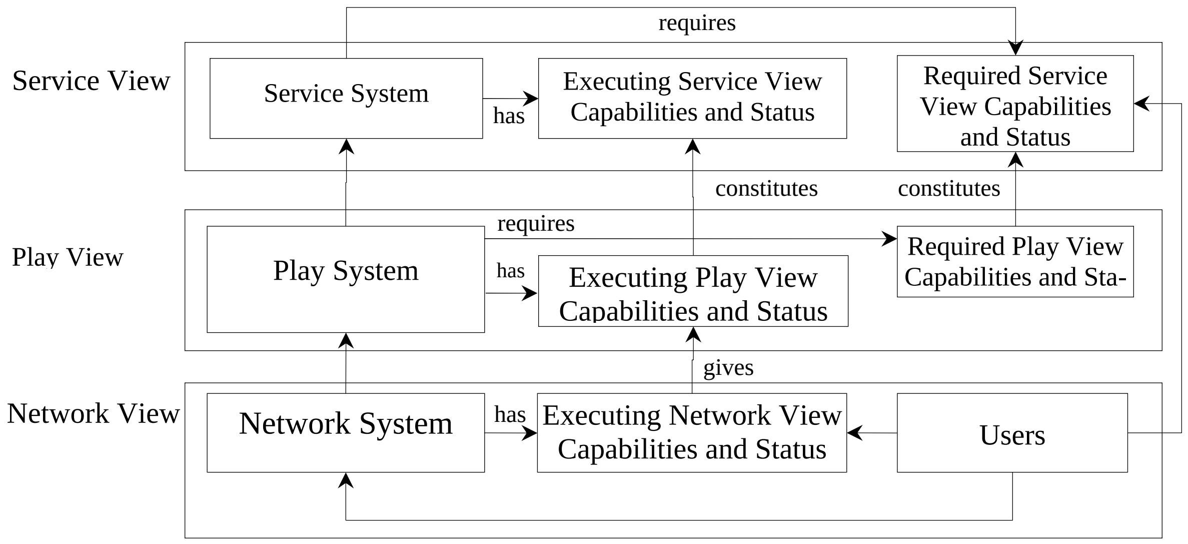

requires
Service View
Service System
has
Executing Service View Capabilities and Status
Required Service View Capabilities and Status
constitutes
constitutes
Play View
requires
Play System
has
Executing Play View Capabilities and Status
Required Play View Capabilities and Sta-
gives
Network View
Network System
has
Executing Network View Capabilities and Status
Users

Figure 2. The TAPAS computing architecture

4.2 System management architecture

The main functionality components of the system management architecture are illustrated in the Figure 3. The primary service providing functionality comprises the ordinary services offered to human users.

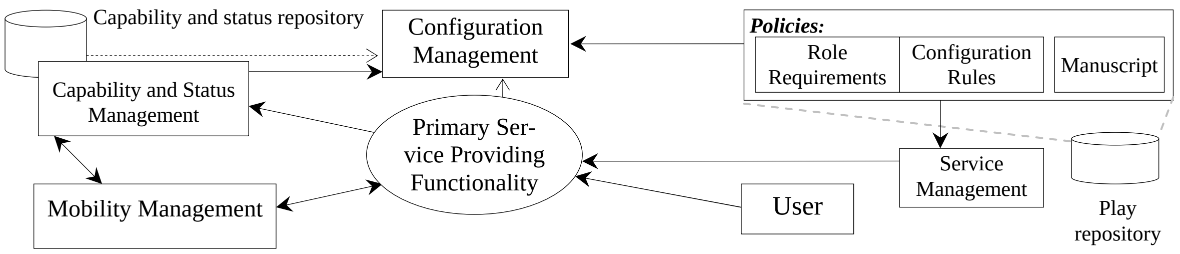


Figure 3. The TAPAS system management architecture

In addition, the architecture has two repositories: the Play repository and the capability and status repository and fours management components: Configuration, Service, Capability and status, and Mobility management.

The play repository stores *manuscripts* and *policies*, which are the required status and capability of a role as well as local configuration rules. Local configuration rules describe configuration and constraints of a role which must always be maintained. In addition, these rules define policies for handling of reconfiguration related events such as the decision of an actor to move a role when a failure happens. The capability and status repository stores *executing capability* and *status information.*

Configuration management makes the initial configuration and re-configures the service systems when needed. The Service management is responsible for deployment and invocation of services. Capability and status management registers, de-registers, updates and provide access to capability and status repository and the Mobility management handles the various mobility types.

To fulfill the failure robustness and survivability requirements, the architecture must be dependable and distributed. The proposed actor model creates a distributed configuration management by adding reasoning functionality to actors.

5. Data model

This section presents XML based approaches to the representation of the elements of the Play repository as well as the Capability and status repository.

5.1 Manuscript

A manuscript consists of EFSM-based behavior of individual roles. An XML-based EFSM given to an actor is executed by a state machine interpreter. A sample fragment of the XML-base manuscript is shown in Figure. 4.

```
<state name='ConnectionTimeout'>...</state>
<state name='ConnectionLost'>
  <Transition name='RoleFigureMove'>
    <input msg='RoleFigureMoveReq' source='*'/>
    <action class='Reasoning' name='SearchFreeActor'>
       <param name='role_name' value='role1'/>
    </action>
    <output><variable name='Dest_Variable'/></output>
    <action class='Communication' name='PluginActor'>
      <param name='actorList' value='Dest_Variable'/>
      <param name='role_name' value='role1'/>
    </action>
    <next_state name='PlugoutPending'/>
  </Transition>
  ...
</state>
```

After the state *ConnectionTimeout* is visited infinitely often, the actor playing this manuscript will move to *ConnectionLost* state. If there is an incoming message *RoleFigureMoveReq*, the actor will execute the *RoleFigureMove* transition and perform two subsequent actions. The first action uses the built-in reasoning machine to find out a free actor where the role should be moved to. The second action installs the role to a free actor suggested by the first action. At the end of the transition, the actor moves to *PlugoutPending* state and wait for a plugout message from the newly instantiated role figure.

Figure 4. Fragment of an example XML manuscript showing a transition of state *ConnectionLost*

SMI interprets the downloaded manuscript. SMI uses *action libraries*. Policy related actions are platform independent constraints expressed in XET (see Section 2). For non-policy actions, the actions are platform-specific (such as C++) or platform-independent (such as Java) executable codes from the local action library cache to execute the actions in the transition. If the required action libraries cannot be found, SMI will download the actions from an action library database.

5.2 Executing Capability and Status

Nodes possess particular Network View Capabilities and Status, from now on abbreviated as NV-capabilities and -status. They are represented in a network information model such as Common Information Model (CIM) or Universal Plug-and-Play. We have chosen the XML representation of CIM (CIM-XML) to implement our test systems.

Actors have Play View capabilities and status abbreviated as PV-capabilities and -status. The idea is to hide the complexity of the network view. PV-capabilities and -status of an actor are derived from one or more NV-capabilities and -status. PV-capabilities and -status are represented in Resource Definition Framework (RDF) [9], which can be used to either define pointers to NV-capabilities and -status or define derived PV-capabilities and -status from NV-capabilities and -status [8].

5.3 Policies

The policies comprises: role requirements, local configuration rules. These are modeled by the XET language (See Section 3).

5.3.1 Role requirements

Role requirements consist of PV-capabilities and -status required by a role. These PV-capabilities and -status are represented in RDF and XML variables.

5.3.2 Local configuration rules

The heads of the XET clauses identify components of the outcome of the configuration or reconfiguration, while the body describes the configuration, composition and dependency conditions. A sample local configuration rule is illustrated in Fig. 5.

```
<xet:Rule name='SearchFreeActor' priority='3'>
   <xet:Head>
    <tapas:Actor rdf:resource='Svar_ActorID'/>
  </xet:Head>
  <xet:Body>
    <xfn:FactQuery xfn:uri='ds://PV-Repository'
xfn:mode='Set'>
      <tapas:Actor rdf:about='Svar_ActorID'>
        <tapas:connectivity rdf:resource='dbServer'>
          <tapas:connStatus rdf:resource='Status_Active'/>
          <tapas:connType rdf:resource='Svar_connType'/>
          Evar_otherConnProps
        </tapas:connectivity>
        <tapas:actorStatus rdf:resource='Status_FreeActor'/>
        Evar_otherActorProps
      </tapas:Actor>
    </xfn:FactQuery>                       Query Expression
    <xfn:StringIsMember xfn:string='Svar_connType'
        xfn:list='Secured SecuredWireless'>
  </xet:Body>
</xet:Rule>
```

Intuitively, this rule looks for free actors that have a secured connection with *dbServer*, which is a database server providing sensitive information. The head of the rule will be derived as answer(s) if both body atoms can be successfully executed.

Namespace *xfn* refers to built-in atoms providing mathematic operations and database query, etc. These atoms will not be further matched with other rules. *FactQuery* queries actors from the capability and status repository. The query expression simply ignores the order of XML elements when it is working in mode *"set"*. Some irrelevant PV-capabilities and -status of actors are ignored by using two E-variables. *Evar_otherConnProperties* and *Evar_otherActorProps*.

The actors must have *Status_FreeActor* as specified in the query expression. They must have only *active secured* or *secured wireless connectivity* with *dbServer1*, which

	% % % will be checked by the builtin atom, *StringIsMember*.

Figure 5. An example XET clause to search for free actors

6. Demonstration

A scenario of a secured database system is considered as an example. A database server contains sensitive information and will automatically blocks incoming requests from nodes that could possibly have malicious software such as viruses or trojans.

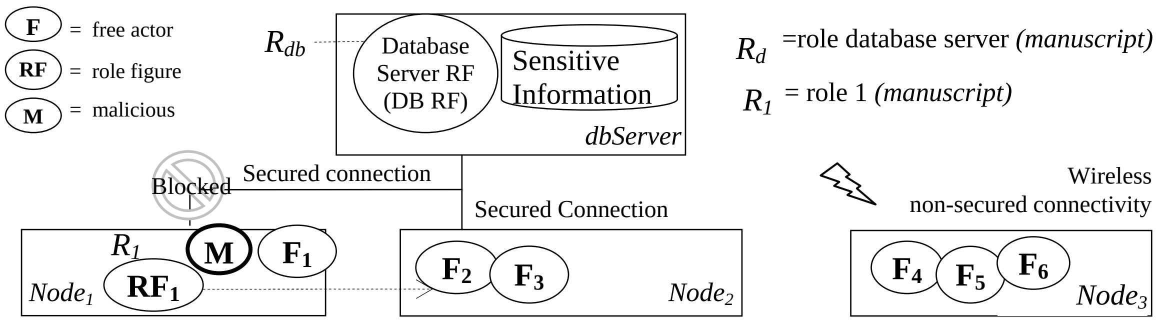


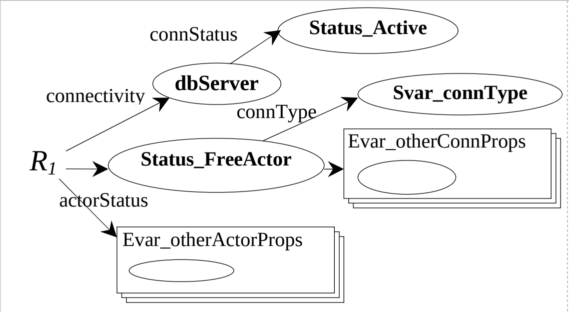
Figure 6. A sample scenario showing the survivability of a role figure.

The goal of this demonstration is to show how a role figure in a blocked node can survive, move to other one other node, identified as harmless by the database server, and continue working with the database server. How the role figure proves itself as non-malicious software is not the focus and will not be further explained.

Fig. 6 illustrates a role figure *RF₁* in Node 1, which is presently blocked by a database server role figure (*DB RF*). After *RF₁* visits a state *ConnectionTimeout* infinitely often, it will move to *ConnectionLost* state. At *ConnectionLost*, *RoleFigureMove* transition will initiated by a *RoleFigureMoveReq* message. The manuscript describing this transition has been presented in Figure 4.

6.1 R1 role requirement (required PV-capabilities and -status)

The PV-capabilities and -status required by the role R_1 are illustrated in Fig. 7. R_1 explicitly needs status *Status_FreeActor*. The connectivity between R_1 and *dbServer* must be a member of set {“Secured”, “SecuredWireless”}. Actors trying to play R_1 may have other PV-capabilities and -status (as represented by *Evar_otherActorProps* and *Evar_otherConnProps*). These PV-capabilities and -status will be ignored by the reasoning engine.



Svar_connType $\in$ **{“Secured”, “SecuredWireless”}**

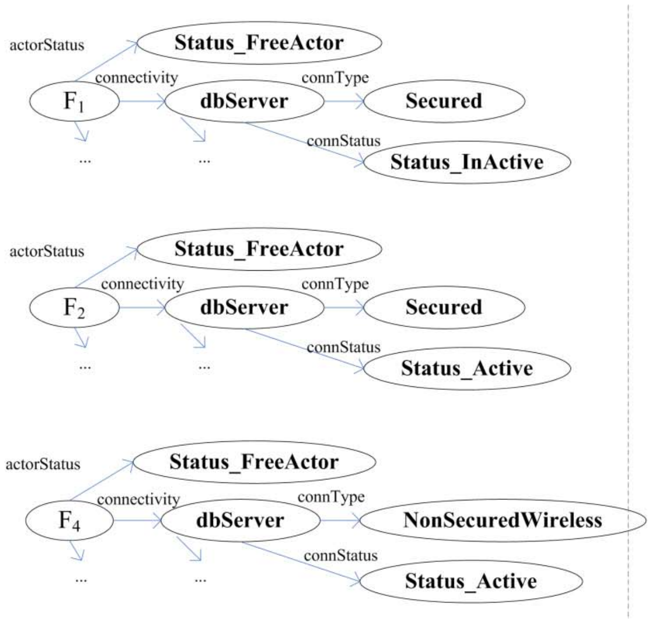
```
<tapas:Role rdf:about='R1'>
  <tapas:connectivity rdf:resource='dbServer'>
    <tapas:connStatus rdf:resource='Status_Active'/>
    <tapas:connType rdf:resource='Svar_connType'/>
   Evar_otherConnProps
  </tapas:connectivity>
  <tapas:actorStatus rdf:resource='Status_FreeActor'/>
  Evar_otherActorProps
</tapas:Actor>


<xfn:StringIsMember xfn:string='Svar_connType'
                              xfn:list='Secured SecuredWireless'/>
```

Figure 7. The required PV-capabilities and -status of the role R1

6.2 Offered PV-capabilities and -status

Fig. 8 shows the offered PV-capabilities and -status of actor F_1, F_2 and F_4. The PV-capabilities and -status of F_3 are identical to F_2 while the capabilities and status of F_5 and F_6 are identical to F_4. For lack of space, they will not be presented.

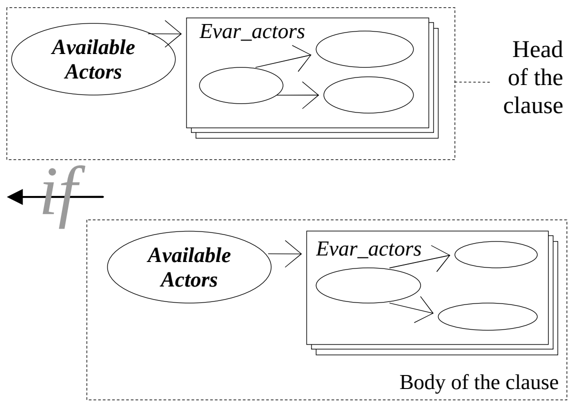


```
<tapas:Actor rdf:about='F1'>
  <tapas:connectivity rdf:resource='dbServer'>
    <tapas:connStatus rdf:resource='Status_Active'/>
    <tapas:connType rdf:resource='Secured'/>
    …
  </tapas:connectivity>
  <tapas:actorStatus rdf:resource='Status_FreeActor'/>...
</tapas:Actor>
<tapas:Actor rdf:about='F2'>
  <tapas:connectivity rdf:resource='dbServer'>
    <tapas:connStatus rdf:resource='Status_Active'/>
    <tapas:connType rdf:resource='Secured'/>
    …
  </tapas:connectivity>
  <tapas:actorStatus rdf:resource='Status_FreeActor'/>…
</tapas:Actor>
<tapas:Actor rdf:about='F4'>
  <tapas:connectivity rdf:resource='dbServer'>
    <tapas:connStatus rdf:resource='Status_Active'/>
    <tapas:connType rdf:resource='NonSecuredWireless'/>
    …
  </tapas:connectivity>
  <tapas:actorStatus rdf:resource='Status_FreeActor'/>…
</tapas:Actor>
```

Figure 8. The offered PV-capabilities and -status of actor F1, F2 and F4

6.3 Query clause

The query clause in Fig. 9 is constructed from a query expression. As already explained in Section 3, the body of the clause will be initially matched with a configuration rule, which will be defined in the Section 5.4.



% The query clause explains that *AvailableActors*
% consisting an unknown set of actors will be derived
% if the reasoning engine can find a rule that matches
% the body of the clause.

```
<xet:Query>
  <xet:QueryClause>
    <xet:Head>
      <tapas:AvailableActors>
        <tapas:consistsOf rdf:parseType='Collection'>
          Evar_actors
        </tapas:consistsOf>
      </tapas:AvailableActors>
    </xet:Head>
    <xet:Body>
      <tapas:AvailableActors>
        <tapas:consistsOf rdf:parseType='Collection'>
          Evar_actors
        </tapas:consistsOf>
      </tapas:AvailableActors>
    </xet:Body>
  </xet:QueryClause>
</xet:Query>
```

Figure 9. Graphical notation of the query to search for available actors

6.4 Local configuration rules

Local configuration rules as illustrated in Fig. 10 indicate that the connection status and the connection type of the link between R_1 and *DB RF* must be maintained in a secured manner. The only QoS parameter defined here is *Svar_connType*.

6.5 The configuration result

The configuration result is shown in Fig 11. Based on the offered PV-capability and -status provided in Fig. 8 and the role requirement defined in Fig. 7, actors F_2 and F_3 are the most appropriate actors to play R_1.

The reasoning process is conducted by Native XML Equivalent Transformation reasoning engine (NxET) implemented as a Java-based action for the state machine interpreter (SMI). NxET is used by SMI to execute *SearchFreeActor* action defined in Fig. 5. The parameter *actorList* of the *PluginActor* action will be substituted with the available actors in Fig. 11. *PluginActor* will try to move R_1 to F_2 first. If the moving is not successful, *PluginActor* will try again with F_3. Subsequently, RF_1 will move to *PlugoutPending* state after R_1 has been successfully moved to either R_2 or R_3. At this state, R_1 will be plugged out from RF_1, which will become a new free actor.

```
<xet:Rule name='SearchFreeActor' priority='3'>
   <xet:Head>
    <tapas:AvailableActors>
      <tapas:consistsOf rdf:parseType='Collection'>
        Evar_actors
      </tapas:consistsOf>
   </tapas:AvailableActors>
  </xet:Head>
  <xet:Body>
    <xfn:SetOf xfn:mode='Set'>
      <xfn:Set>Evar_actors</xfn:Set>
      <xfn:Constructor>
        <tapas:Actor rdf:resource='Svar_ActorID'/>
      </xfn:Constructor>
```

% Rule *SearchFreeActor* will be matched with the
% body of the query clause defined in Section 5.3.
% After the matching, the body of the query clause
% will be re-written with *xfn:SetOf*, which is the only
% body atom of the rule. *xfn:SetOf* will to try to
% construct the list of available actors and add them
% into *Evar_actors* variable. Each members of
% *Evar_actors* will have the structure similar to the
% expression in *xfn:Constructor*. To actually instan-
% tiate the possible values for the constructor, the
% condition expression in *xfn:Condition* will be
% matched with other rules (clearly *R1Requirement*).

```
      <xfn:Condition>
        <tapas:Actor rdf:resource='Svar_ActorID'/>
      </xfn:Condition>
    </xfn:SetOf>
  </xet:Body>
</xet:Rule>
<xet:Rule name='R1Requirement' priority='4'>
   <xet:Head>
    <tapas:Actor rdf:resource='Svar_ActorID'/>
  </xet:Head>
  <xet:Body>
    <xfn:FactQuery xfn:uri='ds://Play-Repository'
xfn:mode='Set'>
      <tapas:Role rdf:about='Svar_RoleID'>
        Evar_properties
      </tapas:Role>
    </xfn:FactQuery>
    <xfn:FactQuery xfn:uri='ds://PV-Repository'
xfn:mode='Set'>
      <tapas:Actor rdf:about='Svar_ActorID'>
        Evar_properties
      </tapas:Actor>
    </xfn:FactQuery>
    <xfn:MatchD xfn:mode='Set'>
      <Expression>
        <tapas:connectivity rdf:resource='Svar_resource'>
          <tapas:connType rdf:resource='Svar_connType'/>
          Evar_otherConnProps
        </tapas:connectivity>
        Evar_otherActorProps
      </Expression>
      <Expression>Evar_properties</Expression>
   </xfn:MatchD>
   <xfn:StringIsMember xfn:string='Svar_connType'
        xfn:list='Secured SecuredWireless'/>
  </xet:Body>
</xet:Rule>
```

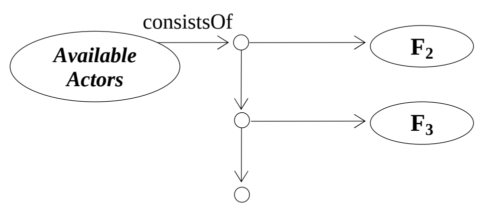
```
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
```

R1Requirement queries the R1 role requirement (required PV-capabilities and -status), which have been defined in Section 5.1. The rule again queries actors offering the same PV-capabilities and -status, which R_1 requires. The matching between required and offers PV-capabilities and -status are accomplished though the instantiation of variable *Evar_properties*. The actors queried from the capability and status repository needs *Status_Active* and *Status_FreeActor*. The actors can also have other PV-capabilities and -status because they are allowed by the role requirement.

The structure of PV-capabilities and -status of each actor will be matched with *xfn:MatchD* function so that PV-capability *connType* with a value *Svar_connType* can be inspected. Function *StringIsMember* verifies the instantiated value of *Svar_connType* to make sure that it is a member of the list *"Secured SecuredWireless"*. Actors that do not offer secured or secured wireless connection will be filtered out. Only qualified one will be selected. *R1Requirement* can return many answers.

The answers returned by *R1Requirement* will be aggregated and added to *Evar_actors* list in the rule *SearchFreeActor*. The value of *Evar_actors* will be instantiated to the head of the query clause, which will be the answer of the reasoning process.

Figure 10. Local configuration rules in XET



```
<tapas:AvailableActors>
  <tapas:consistsOf rdf:parseType='Collection'>
     <tapas:Actor rdf:resource='F2'/>
     <tapas:Actor rdf:resource='F3'/>
  </tapas:consistsOf>
</tapas:AvailableActors>
```

Figure 11. RDF-based graphical notation and XML-serialization of the configuration result

7. Conclusion

This paper presents an approach to model the behavior of service systems by actors playing roles defined in manuscripts. The actor is a combination of an Extended Finite State Machine (EFSM) and a rule based reasoning engine.

A service system has defined requirements to capabilities and status. Because of continuous changes in capabilities and status, dynamic service configuration with respect to capabilities and status is needed. Configuration is based on the matching between required capability and status of a role and the present executing capabilities and status. Roles are allowed to be moved to increase failure robustness and survivability of a service system. This role mobility can be achieved through EFSM behavior. However, using a rule-based reasoning mechanism allows actors to use local configuration rules to take decisions based on the current executing capabilities and status. The actor model improves actor functionality, increases survivability and makes the configuration management distributed.

Generic actor execution support consisting of a state machine interpreter and a reasoning engine has been implemented and applied for the presented example. All capability and status related data as well as actor behavior is based on XML representations, with exceptions of the EFSM actions. Normal EFSM actions are platform-specific (such as C++) or platform-independent (such as Java) executable codes while reasoning-based EFSM actions are XML-based. The reasoning engine is based on Native XML Equivalent Transformation.

References

[1] Aagesen, F.A., et al., Configuration Management for an Adaptable Service System, *IFIP Open Conference on Metropolitan Area Networks Architecture, protocols, control, and management*, Viet Nam, 4/2005

[2] Aagesen, F.A., et al., On Adaptable Networking. *ICT'2003, Assumption University*, Thailand, 4/2003.

[3] Anutariya, C., et al., An Equivalent-Transformation-Based XML Rule Language. *Int'l Workshop Rule Markup Languages for Business Rules in the Semantic Web*, Italy, 6/2002.

[4] Bonino, D., et al., An agent based autonomic semantic platform, *Proc. Int'l Conf. on Autonomic Computing 2004*, 5/2004.

[5] Huang, G., et al., Towards autonomic computing middleware via reflection, *Proc. of the 28th Annual International COMPSAC 2004*. 9/2004.

[6] Inoue, Y., et al., The TINA Book. A Co-operative Solution for a Competitive World. Prentice Hall, 1999.

[7] Shiaa, M.M., Mobility Support Framework in Adaptable Service Architecture. *IEEE/IFIP Net-Con'2003*, Oman, 10/2003.

[8] Supadulchai, P., Aagesen, F.A., An Approach to Capability and Status Modeling, *NIK 2004*, Norway, 11/2004.

[9] World Wide Web Consortium, Resource Description Framework (RDF): Concepts and Abstract Syntax, Available online at http://www.w3.org/TR/rdf-concepts/.

Paper E

Policy-based Adaptable Service Systems Architecture

Paramai Supadulchai and Finn Arve Aagesen

In Proceedings of *the IEEE 21st International Conference on Advanced Information Networking and Applications* (AINA-07)

Niagara Falls, Canada, May 21-23, 2007

Policy-based Adaptable Service Systems Architecture

Paramai Supadulchai and Finn Arve Aagesen

Department of Telematics
Norwegian University of Science and Technology (NTNU)
N7491 Trondheim, Norway
paramai@item.ntnu.no, finnarve@item.ntnu.no

Abstract

This paper presents a policy-based architecture for adaptable service systems based on the combination of Reasoning Machines and Extended Finite State Machines. Policies are introduced to obtain flexibility with respect to specification and execution of adaptable service systems that give high performance over a range of system status values. The presented architecture covers three aspects: service system framework, adaptation mechanisms and data model. The adaptation mechanisms can be based on static or dynamic policy systems. Static policy systems have a non-changeable set of policies, Dynamic policy systems have a changeable set of policies, which are managed by policies at a higher level. The data model for the reasoning machine functionality is based on the rule-based reasoning language "XML Equivalent Transformation" (XET). The capability configuration management of a service system with runtime simulation results based on the proposed architecture is presented with the intention to illustrate the use of the architecture and discuss the potential advantages of using dynamic policies.

1. Introduction

Networked service systems are considered. Services are realized by the structural and behavioral arrangement of service components, which by their interworking provide a service in the role of a service provider to a service user. Service components are executed as software components in nodes, which are physical processing units such as servers, routers, switches and user terminals.

An adaptable service system is a service system which is able to adapt dynamically to changes in time and position related to users, nodes, status of capability and service performance measures, changed service requirements and policies.

A capability is an inherent property of a node or a user, which defines the ability to do something. Capabilities can be classified into resources, functions and data. A capability in a node is a feature available to implement services. Examples are CPU, memory, transmission capacity of connected transmission links, available special hardware, and available programs and data. A capability of a user is the feature that makes the user capable of using services. Capability performance measures are the set of variables used for the performance monitoring and management of capabilities. Capability performance measures reflect the capabilities with respect to being idle or allocated, available capacity, load etc.

Service performance measures are the set of variables used for the performance monitoring and management of the operation of the service system. These can be measures reflecting the state of a service system with respect to the number of active service components as well as Quality of Service (QoS) measures. Capability and service performance measures represent performance measures in two different viewpoints: the network view, which considers the concrete physical elements and capabilities and the service view, which consider the abstract service elements constituting the service.

Status is the present value of a capability (capability status) or a service performance measure (service status). The total set of status measures is denoted as system status, which then is the sum of capability status and service status. Service components will have requirement with respect to system status.

The software mechanisms used for implementing the functionality of the service components of adaptable service systems must be flexible and powerful. Service components based on the classical EFSM (Extended Finite State Machine) approach can be flexibly executed by using generic EFSM executing software components that are able to download and execute different EFSM based specifications. The TAPAS architecture [1] has, in addition to a service and network view defined above, a play view that realizes this feature. In TAPAS the generic software components are denoted as Actors, inspired by the actors in the theatre. Actors are able to play various roles specified in EFSM-based manuscripts that are dynamically downloaded.

In addition to this type of flexibility constituted by Actors that are able to play various EFSM-based specifications, the EFSM-based functionality can be supplemented by Reasoning Machine (RM)-based functionality, which makes policy-based specification and operation possible. “Policies represent externalized logic that can determine the behavior of the managed systems” [2]. In this paper a policy is technically defined as a set of rules with related actions. A policy system is a set of policies, and an RM-based functionality is using a policy system to manage the behavior of a target system. A static policy system has a non-changeable set of rules and actions, while a dynamic policy system has a changeable set of rules and actions. The target system for a policy system can be another policy system, so that one policy system can be controlled by another policy system. The dynamic policy system is managed by control rules and actions constituting a policy system at a higher level.

The EFSM approach needs careful specification of all possible events. Policy-based software is based on rules, and has a specification style, which is expressive and flexible. Human defined procedures at both business and customer service levels are often more easily expressed as policies rather than expressed as EFSMs. This is because the expressiveness and the flexibility of rules are often more directly applicable than EFSMs. In addition to being used at business management and customer service provision levels, rules can also be at the service system and service component level as a supplement to the use of EFSMs. Software functionality based on policy-based specifications, however, also needs to be appropriately specified and validated. We are not claiming that validation is easier for policy-based specification than for EFSM-based specifications. The validation aspect, however, is outside the scope of this paper.

In general, adaptation needs appropriate mechanisms to guarantee the wanted results. Stable feedback loops [3], which control the performance, are needed. As the capabilities are limited, an adaptable system needs to limit the access to the system, and there must also be priority mechanisms that give priority to users which are willing to pay more and/or are in a higher need in situations with lack of capabilities. Policy-based adaptable systems can be based on static or dynamic policies. In the static case, the feedback loop is related to the appropriate rules and actions. In the dynamic case, the system also must have some feedback mechanism related to policy selection. As a basis for the policy selection, the system must have goals, goodness criteria and also the ability to estimate or evaluate the consequences of the use of a policy.

The issues of policy-based adaptable service system architecture are in this paper classified into 3 main aspects: A) service system framework, B) adaptation mechanism, C) data model.

The *service system framework* issues can be classified further as: 1) general component structure, 2) application domain for the RM functionality, 3) the integration of the RM functionality in the component structure, and 4) reasoning procedure. The *reasoning procedure* is the procedure applied by RM to select actions to be applied. In the context of this paper the application domain for the RM functionality is classified as follows:

i) A traditional procedural service for an EFSM-based service components to take decisions not involving the management of system status,
ii) The initial capability configuration of the service system according to the capability status requirements of the EFSM components of a service system,
iii) The adaptation of the behavior of service systems and/or EFSM-based service components involving the management of system status. This also includes capability reconfiguration based on capability and service status requirements,
iv) The dynamic change of the policies of the policies used for the items i)-iii) above

The *adaptation mechanism aspect* (B) concerns the use of the appropriate policies to control the service systems when it is entering a state where RM functionality is needed. The *data* model *aspect* (C) concerns all representation of data and functionality related to the RM functionality.

The papers [1] and [4], have focus on the aspect C) applied on the issues i) and ii) of the application domain for reasoning only. This paper comprises all issues of the aspects A)-C) as defined above.

Section 2 discusses related work. Section 3 presents a model for a service system framework, Section 4 discusses policy-based adaptation mechanism and Section 5 presents the data model used for the RM-based functionality. Section 6 presents four scenarios related to capability configuration management of a music video on-demand service to illustrate the use of the proposed policy-based service system architecture, and the potential advantages of using dynamic policies. Section 7 gives summary and conclusions.

2. Related work

Most of recent works related to policy-based adaptable service systems focuses on the aspects A) and B) as defined in Section 1. Examples are [5], [6], [7], which are Garlan et al.'s Rainbow architecture for self-adaptation, Samaan and Karmouch's autonomous policy-based management framework and Narsi et al.'s learning techniques. A work that support the aspect C), PMAC by Agrawal et al. [2], uses Autonomic Computing Policy Language (ACPL) as a generic data model, which is analogous to our XML Equivalent Transformation (XET). However, this work has a weak focus on the aspects A) and B).

Considering the application domain for the use of policies, the application domain of [5] is preliminary aimed at static policies. The application domains of [6] and [7] are both static and dynamic policies.

The architecture presented in this paper has the focus on all aspects A)-C). The service system framework permits the combination of both EFSM and the RM functionality. Considering the dynamic policy, i.e. the rule-based modification of the policy managing the service system (See Sections 0), the system's policy can be composed at run-time based on *evaluation criteria*, *reference inputs* and *feedbacks*. Unlike [6] and [7], the presented architecture evaluates and composes the best policy based on broad set of *evaluation criteria*, which can be history-based, prediction-based, or logic-based. *Income functions* are used as *reference inputs*, while the *feedbacks* are *system performance measures*.

3. Service System Framework

3. 1 General component structure

An executing service system consists of executing service components. An executing service component is an instance of a *service component type*, which in the context of this paper is modeled by some combination of *EFSM type* and *RM type*. A service component can be a pure EFSM or some combination of EFSM and RM based functionality. The main role of the EFSM functionality is to maintain the state of the service system represented as EFSM states and variables (see below), while the main role of the RM functionality is to take decisions. The interaction between the RM functionality and the EFSMs will be discussed in Section 3.2.

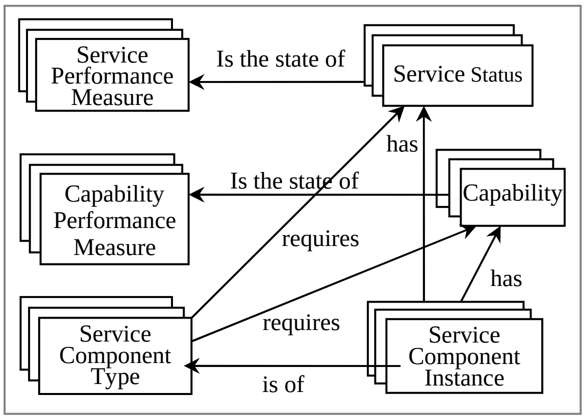


Figure 1 - Service System – General Component

The service components will have requirements with respect to capabilities and capability status to be able to perform their intended functionality (Figure). As a basis for the optimal adaptation, *service level agreements* are needed between the *service users* and the *service provider*. The service provider view of this service level agreement can in this context be considered as a part of an executing service component. A number of QoS priority levels can exist. The agreement can contain elements such as: 1) agreed QoS level, required capabilities, 2) required service status 3) payment for the service in case of normal service and 4) payment for the service in case of reduced service.

The following concepts are defined:

E	Functionality set of an EFSM type
Ê	Functionality set of an EFSM instance
$\mathcal{R}$	Functionality set of a RM type
$\hat{\mathcal{R}}$	Functionality set of a RM instance
C	Capability performance measures set

$\hat{C}_R$	Required capability status set for an EFSM based service component type
$\hat{C}_I$	Inherent capability status set of an executing EFSM based service components
$\hat{C}_A$	Status set of available capabilities in nodes
S	Service performance measures set
$\hat{S}_R$	Required service status set for an EFSM based service component type
$\hat{S}_I$	Inherent service status set of an executing EFSM based service component
I	Income functions set for the service components constituting a service.

The set of requirements related to capability and service performance measures are denoted as *required capability and service status,* respectively. The status of capabilities and service performance measures of the executing system are similarly denoted as *inherent capability and service status.* The income functions will depend on the system status.

An EFSM type E is defined as:

$$\mathrm{E} \equiv \{ S_M, S_I, V, P, M(P), O(P), F_S, F_O, F_V \}, \quad (1)$$

where S_M is the set of states, S_I is the initial state, V is a set of variables, P is a set of parameters, $M(P)$ is a set of input signal with parameters, $O(P)$ is a set of output signal with parameters, F_S is the state transition function ($F_S = S \times M(P) \times V$), F_O is output function, ($F_O = S \times M(P) \times V$) and F_V are the functions and tasks performed during a specific state transition such as computation on local data, communication initialization, database access, etc.

A RM type $\mathcal{R}$ is defined as:

$$\mathcal{R} \equiv \{ \mathcal{Q}, \mathcal{F}, \mathcal{P}, \mathcal{T}, \mathcal{E}, \Sigma \} \quad (2a)$$
$$\mathcal{P} \equiv \{ \mathcal{X}, \mathcal{A} \}, \quad (2b)$$

where $\mathcal{Q}$ is the set of messages, $\mathcal{F}$ is a generic *reasoning procedure*, $\mathcal{P}$ is a policy system which consists of a set of rules $\mathcal{X}$ and a set of actions $\mathcal{A}$, $\mathcal{T}$ is a set of *system constraints* and $\mathcal{E}$ is a set of *status data*. The status data represents the status of the variables of the targeted system. The system constraints represent the variables of the system and the defined constraints and relationships between variables. The policy rules are based on the variables of the constraints. Σ is a set of reasoning conditions, which define the conditions for the use of RM functionality. The reasoning condition set consists of: *trigger conditions Σ_T, and goal conditions Σ_G.* RM functionality is activated when a Σ_T is detected until a Σ_G is reached. Assuming that a trigger condition is true, the reasoning procedure transforms $\mathcal{Q}_i$ to $\mathcal{Q}_j$ by using $\mathcal{P}$ to match the system constraints $\mathcal{T}$ against the *status data* $\mathcal{E}$ and a set of suggest actions $\{\mathcal{A}_i, \mathcal{A}_j, \mathcal{A}_k \ldots \} \subseteq \mathcal{A}$. These actions may also set the next state and values of the variables of EFSM-based service component instances.

The equations (1), (2a) and (2b) describe a generic model. The concrete modeling of RM-based functionality will be based on this model.

3.2 Reasoning machine based functionality – application domain and integration

The application domain of the RM functionality can be defined by the four cases: i) – iv) as classified in Section 1. In Case i), an RM is a supplement to the tasks (F_V) of an EFSM. In the second case an RM determines the capability configuration of the EFSMs constituting the service system. In the third case an RM influences the behavior of the service system, but with static policies. In the fourth case an RM is used as a policy- based adaptation mechanism for the policies to be selected.

In addition to having RM functionality as a supplement to the EFSM-based service system, the RM functionality will also need EFSM support for the continuous updating of the system constraints, status data and reasoning conditions: $\mathcal{T}$, $\mathcal{E}$ and Σ, as well as the activation and deactivation of the reasoning machine based on the present value of Σ. The continuous updating of $\mathcal{T}$, $\mathcal{E}$ and Σ is done by EFSMs and in this case the $\mathcal{T}$, $\mathcal{E}$ and Σ is considered as common data for the EFSMs and the associated RM based functionality. A dedicated EFSM associated with the RM-based functionality denoted as E_{Σ} has the duty to inspect the reasoning condition and to activate and to deactivate the reasoning machine.

3.3 Reasoning procedure

The *Reasoning Procedure* is the procedure applied by the reasoning machine to select the rule to be applied. The *Reasoning procedure* is based on *Equivalent Transformation* (*ET*) [8], which solves a given problem by transforming it through repetitive application of (semantically) equivalent transformation rules.

ET consists of sets of *ET rules* and ET *clauses*. A problem must be formulated as a clause for transformation. An ET *clause* has the form:

$$\underbrace{\text{Head atom}}_{\textbf{Head}} \longleftarrow \underbrace{\text{Body atom}_1, \ldots \text{Body atom}_n}_{\textbf{Body}}$$

Head of a clause consists of an atom, or the *head atom*, which is a message containing a problem with unknown answer(s)/action(s). The problem in the head atom will be derived by *rules* until it eventually contains a list of suggested action(s).

An *ET rule* has the form:

$$\underbrace{\text{Head atom}}_{\textbf{Head}} \quad \underbrace{\text{Condition atom}_1, \ldots}_{\textbf{Condition}} \longrightarrow \underbrace{\text{Body atom}_1, \ldots \text{Body atom}_n}_{\textbf{Body}}$$

It consists of a rule head and a rule body. A body atom of a clause matching the head atom of a rule can be transformed into the rule's bodies. A *rule of a policy* as defined in Section 3.1 is modeled by an *ET rule*

Intuitively, the reasoning procedure begins with a clause formulated by a message as follows:

$$\boldsymbol{msg(\ldots)} \longleftarrow \boldsymbol{msg(\ldots)} \qquad (3)$$

The meaning of (3) is that the head *msg*(…) is true when the body *msg*(…) is true, which we don't need to prove. The goal of the reasoning procedure is to transform (3) until no body atom is left. Consider the following rule (4):

$$msg(\ldots),\ \mathrm{C} \xrightarrow{ET} B_1, B_2, \ldots B_n. \qquad (4)$$

The rule (4) can transform the body atom msg(…) of (3) into B_1, B_2, … B_n; provided that the atom msg(…) match the head of (4) and the set of conditions C is not violated. Clause (3) will be transformed to (5) as follow:

$$msg(\ldots) \longleftarrow B_1, B_2, \ldots B_n. \qquad (5)$$

During the transformation, variables in msg(…) including the *unknown list of suggested actions*, which are a subset of the actions $\mathcal{A}$ as defined in (2.b), will be instantiated. The transformation of a clause ends when either 1) there is no body atom left or 2) there is no rule that can transform the remaining body atoms.

4. Policy-based adaptation mechanism

4.1 System constraints, status data and reasoning conditions

The elements $\mathcal{T}$ and $\mathcal{E}$ of an RM as defined in Section 0, depend on the structuring and the nature of the reasoning functionality. They depend on which EFSMs that are related to the RM functionality and also the nature of the reasoning. A *reasoning cluster*, *which* is an independent unit with respect to reasoning, is a collection of EFSM-based service components with an associated *reasoning system* constituted by one or more *reasoning machines*. A reasoning cluster has a set of associated income functions I. The elements $\mathcal{T}$ and $\mathcal{E}$ of a reasoning cluster with available capabilities from N_{Node} nodes, consisting of K EFSM-based service component types and L_k instances of an EFSM-based service type k are defined as follows:

$$\mathcal{T} \equiv \mathit{Expr}\,\{S, C, I, (E_k, \hat{S}_{R,k}, \hat{C}_{R,k}; k = [1,K])\} \quad (6)$$

$$\mathcal{E} \equiv \{((\hat{E}_{lk}, \hat{S}_{I,lk}, \hat{C}_{I,lk}; l = [1, L_k]), k = [1, K]), (\hat{C}_{A,n}; n = [1, N_{Node}])\} \quad (7)$$

The function $\mathit{Expr}\{X_i;\ i = [1, I]\}$ in (6) symbolizes the set $\{X_i;\ i = [1, I]\}$ and also some set of logical functions based on the elements of the set. The system constraints $\mathcal{T}$ related to a reasoning cluster comprise the EFSM functionality sets of the EFSM-based service component types, required capability and service status, as well as the income functions for the reasoning cluster. The status data $\mathcal{E}$ defined in (7) is a set of the inherent capability and service status for all instances of EFSM- based service components in the reasoning cluster, as well as available capabilities of the nodes that potentially can contribute their capabilities for the EFSM based functionality of the reasoning cluster.

As stated in Section 1 service system adaptation also includes capability reconfiguration based on capability and service status requirements. *Capability configuration management* is the service systems initial capability configuration and reconfiguration. Capability configuration management goes beyond the boundaries of an individual reasoning clusters as well as an individual service system. This means that capability configuration management must be handled by a common distributed algorithm or by a centralized reasoning cluster.

The components constituting the *reasoning condition* Σ are the states and variables of $\hat{E}$, and the capability and service performance measures C and S as given in (8).

$$\Sigma \equiv \mathit{Expr}\,\{S, C, (E_k, \hat{S}_{R,k}, \hat{C}_{R,k}; k = [1, K])\} \quad (8)$$

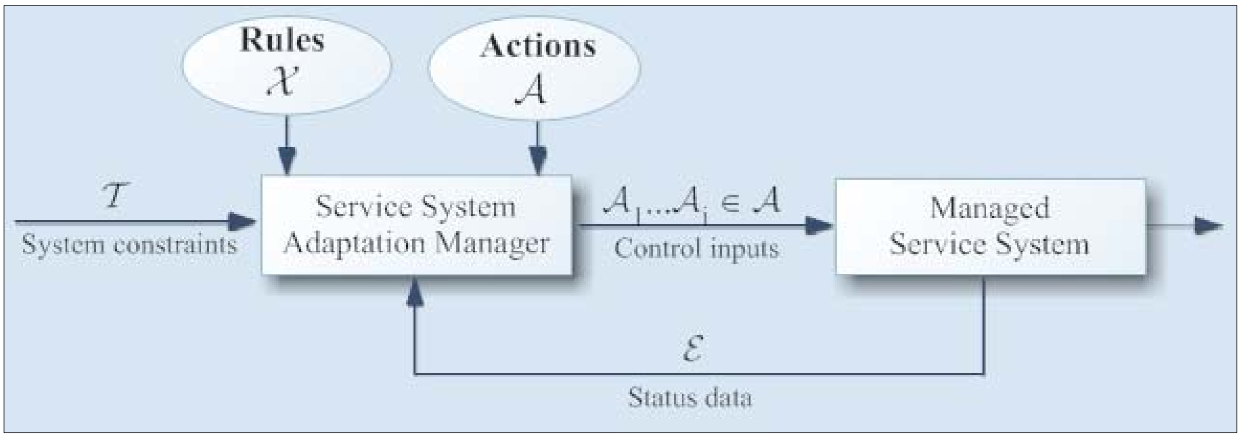


Figure 2 – Policy-based adaptation using static policies

4.2 Policy-based adaptation using static policies

The policy-based adaptation using static policies, as illustrated in Figure 2, manages the behavior of service systems based on system constraints $\mathcal{T}$. The managed service system give its status data $\mathcal{E}$ to the Service System Adaptation Manager $\mathcal{R}_1$, which is a reasoning machine. $\mathcal{R}_1$ gives a set of control inputs $\mathcal{A}_1 \ldots \mathcal{A}_i \subset \mathcal{A}$ (as already defined in Section 3.1) back to the managed service system. A policy system consists of static rules, which is unchangeable. Upon service systems enter a Σ_T, $\mathcal{R}_1$ is activated and try to lead the system back to a goal state Σ_G. $\mathcal{R}_1$ is de-activated when service systems enter Σ_G.

4.3 Policy-based adaptation using dynamic policies

The use of static policies has some disadvantages. Firstly, it has only the priority mechanism to select a rule when two or more rules are applicable and it is hard to assign appropriate priorities to rules, especially when the rule space is large. Secondly, a set of static rules will not likely solve the problem under all set of different conditions that an adaptable service system must handle.

A possible approach is to use dynamic policies as illustrated in Figure 3, and using two reasoning machines. In addition to the Service System Adaptation Manager ($\mathcal{R}_1$) a Policy Composer ($\mathcal{R}_2$) is used. A generic rule-based reasoning system with dynamic policy can be defined by (9a, 9b, 9c and 9d) as follows:

$$\mathcal{R}_1 \equiv \{ \mathcal{Q}, \mathcal{F}, \tilde{\mathcal{P}}, \mathcal{T}, \mathcal{E}, \Sigma \} \tag{9a}$$

$$\tilde{\mathcal{P}} \equiv \{ \tilde{\mathcal{X}}, \tilde{\mathcal{A}} \} \tag{9b}$$

$$\mathcal{R}_2 \equiv \{ \mathcal{Q}', \mathcal{F}, \mathcal{P}', \mathcal{T}', \mathcal{E}', \Sigma' \} \tag{9c}$$

$$\mathcal{P}' \equiv \{ \mathcal{X}', \mathcal{A}' \} \tag{9d}$$

where $\mathcal{T}' = \{I, \mathcal{X}, \mathcal{A}\}$ and $\mathcal{E}' = \{ \hat{C}_I, \hat{S}_I \}$. $\mathcal{Q}'$ is a set of messages between $\hat{\mathcal{R}}_1$ and $\hat{\mathcal{R}}_2$. $\mathcal{X}'$ is a set of control rules that can re-order the priority of the rules, activate and deactivate the rules and change rules' constraints. The policy composer composes the system policy at runtime based on *evaluation criteria*, *reference inputs* and *feedbacks*. Evaluation criteria can be history-based, prediction-based and logic-based. Income functions are used as reference input, while the feedbacks are system performance measures.

The history-based evaluation method determines the consequences of the rules in the past. The prediction-based evaluation determines the consequences of rules in the future based on mathematical equations represented by $\mathcal{X}'$. The logic-based

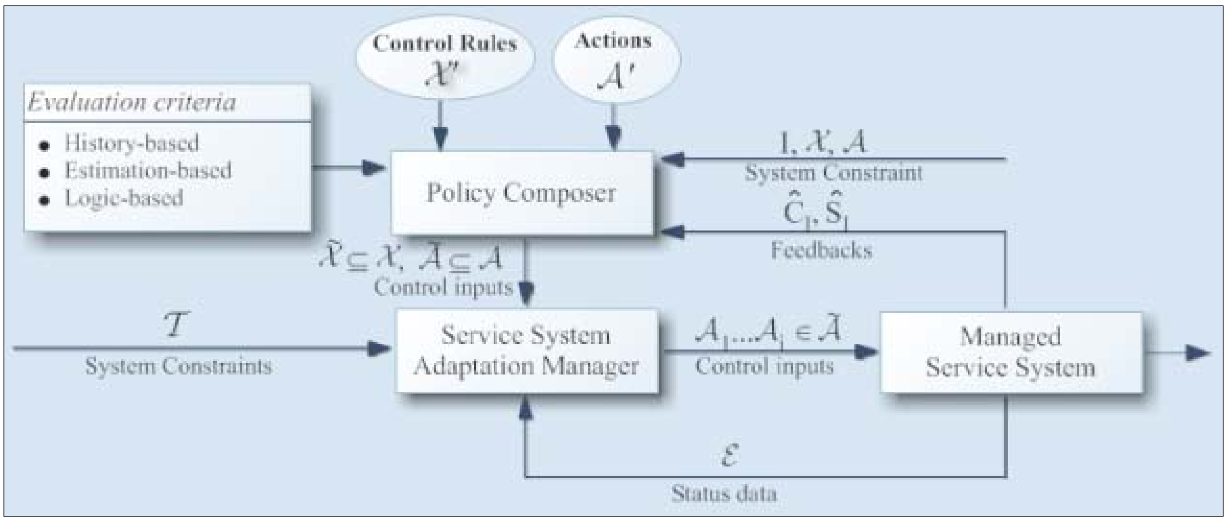


Figure 3 - Policy based adaptation based on dynamic policies

evaluation determines the consequences of rules based on logics such as fuzzy, business-level or user-defined logics represented by $\mathcal{X}$.

5. Data Model

Data Model for the reasoning functionality is based on XML Equivalent Transformation language (*XET*), which is the XML representation of the ET as given in 0. XET represents status data, system constraints, and rules by using Extensible Markup Language (XML), Resource Definition Framework (RDF), XML expressions and XML rules.

XML and RDF: $\hat{C}_I, \hat{S}_I$ and $\hat{C}_A$ are expressed in the XML version of Common Information Model (CIM) denoted as xmlCIM [1]. RDF is used because CIM does not provide means for representing $\mathcal{Q}$ required by the developed framework [1]. $\hat{E}_I$ is modeled by the XML-based EFSM proposed by [9].

XML Expressions: XML expressions are used for the data representation of the system constraints $\mathcal{T}$. XML expressions [3] are ordinary XML elements with possible six disjoint variable types . XML expressions can represent implicit information by expressive XML variables. These variables can be specialized (or instantiated) into attributes names, element names, strings, zero or more attribute-value pair(s), one or more XML expression(s) and parts of XML expressions depending on their types (see [10]). An ordinary XML element (or a ground XML expression) is an XML expression without variables.

XET rule: An *XET* rule is the representation of an *ET* rule in XML. The structure for an XET rule is illustrated as follows.

```
<xet:Rule xet:name="…"
xet:priority="…" xet:class="…">
   <xet:Meta>…</xet:Meta>
   <xet:Head>…</xet:Head>
   <xet:Condition>…</xet:Condition>
   <xet:Body>
      Body atom1, Body atom 2, …
   </xet:Body>
</xet:Rule>
```

A head, condition or body atom is represented by a fragment of XML. For example, the body <xet:Body><a1/><a2/></xet:Body> contains two body atoms, which are <a1/> and <a2/> respectively. The xet:Meta contains additional metadata for rules that will be used by control rules.

6. Application examples

6.1 The scenarios

Four scenarios handling the capability configuration management of a *music video on-demand service* are presented with the intention to *illustrate* the *use* of the proposed policy-based service system architecture, and the *potential advantages* of using dynamic policies. Scenario I and II use no policy. Scenario III uses static policies, while Scenario IV uses dynamic policies. The service system is constituted by one or more media servers (MS) streaming media files to media players (MP) (Figure 4). The numbers of MS used in Scenario I and II are fixed (one and two respectively), while the number in Scenario III and IV can vary from one to two. An MP belongs to a *QoS_Class*. In the example two classes are applied: premium (MP_P) and ordinary (MP_O).

Three different streaming throughput bit-rates (X) are offered, 500Kbps, 800 Kbps and 1Mbps. MP_O connections are 500Kbps (X_O) while MP_P connections can be either 800Kbps or 1Mps (X_P).

The *capability performance measures* used are MS access link capacity

(C_{AL}). The required and inherent MS capability status sets are defined as follows:

$$\hat{C}_R \equiv \{ C_{R,AL} \} \quad (10)$$

$$\hat{C}_I \equiv \{ C_{I,AL} \} \quad (11)$$

where $C_{R,AL}$ is the MS's required access link capacity, which is set to 100 Mbit/s.

The number of MPs that can use the service, in this example, is limited by the MS access link capacity. The service level agreements comprise *maximum waiting time*, *required streaming throughput*, *payment for the service* and *penalties for not satisfying the service*. The maximum waiting time for MP_P and MP_O are 60 seconds and infinite respectively. The required streaming throughput of MP_O and MP_P are X_P and X_O respectively. The payment for the service and the penalties for not satisfying the service are calculated by income and penalty functions that will be defined.

The resource management mechanisms used by the service provider is to disconnect ordinary clients, to decrease the throughput of the premium clients and to change the number of media servers.

When the required streaming throughput cannot be provided, an MP may have to wait until some connected MPs have finished using the service. This will result in money payback to the waiting MPs. An MP_O can be disconnected, while an MP_P may have to reduce the throughput. If a client is disconnected, the service provider pays a penalty. If the throughput is lowered, the price is lowered.

The service performance measures $\hat{S}_I$ consists of the number of connected and waiting premium and ordinary clients ($N_{Con,P}$, $N_{Con,O}$, $N_{Wait,P}$, $N_{Wait,O}$), the number of disconnected MP_O ($N_{Dis,O}$), the number of MS (N_{MS}), inherent streaming throughput (X_I), the number of available nodes (N_{Node}) and the accumulated service time and waiting time of premium and ordinary clients ($T_{Serv,P}$, $T_{Serv,O}$, $T_{Wait,P}$, $T_{Wait,O}$). These values are observed per a monitoring interval Δ.

A *cost unit* is the price paid by an ordinary customer for *one second streaming* of the rate 500 KBit/s. The income function for the service provider is m(QoS_Class, X_I) (cost units/second). The penalty function for waiting is p_{Wait}(QoS_Class) (cost units/second). The penalty function for disconnections is p_{Dis}(QoS_Class) (cost units/disconnection). The cost function for adding a new server is p_{Ser} (cost units per Node per sec). The total income function

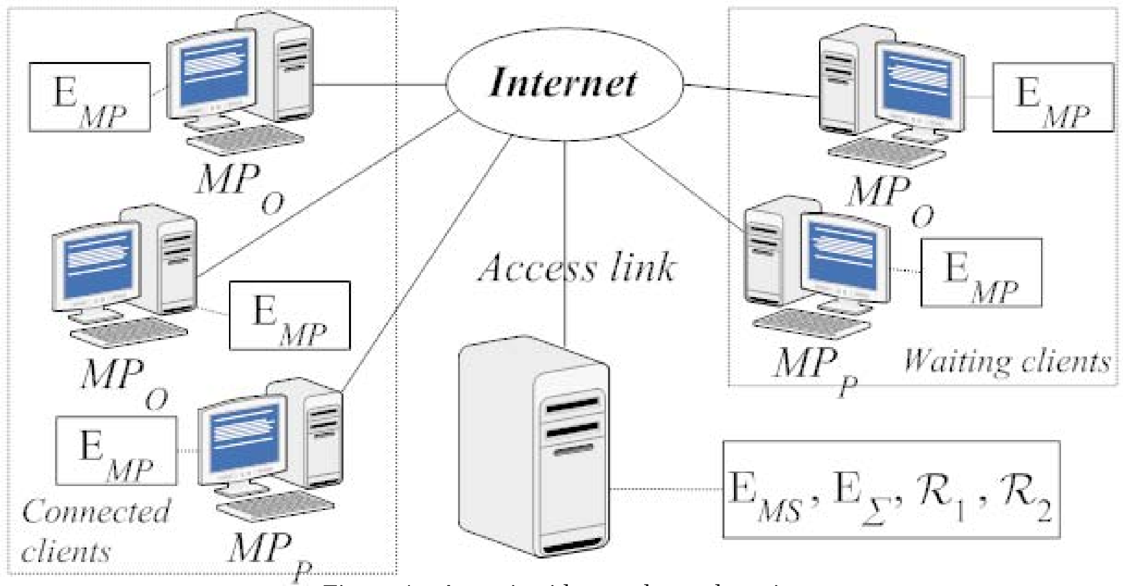


Figure 4 – A music video on-demand service system

(m_T) during the monitoring interval Δ is defined as follows:

$$m_T = m(MP_O, X_{I,O})\times T_{Serv,O} + m(MP_P, X_{I,P})\times T_{Serv,P} - p_{Wait}(MP_O)\times T_{Wait,O} - p_{Wait}(MP_P)\times T_{Wait,P} - p_{Dis}(MP_O)\times N_{Dis,O} - p_{Ser}\times(N_{MS}\text{-})\times\Delta \quad (12)$$

Policy-based adaptation is introduced to maximize the total income. The service system is realized as one reasoning cluster. As illustrated in Figure 4, E_{MS}, E_Σ, $\mathcal{R}_1$ and $\mathcal{R}_2$ are in the same node. E_{MS} is the *media server* type, E_{MP} is the *media player* type, $\mathcal{R}_1$ is the *service system adaptation manager* type and $\mathcal{R}_2$ is the *policy composer* type according to the definitions in Section 4.2 and 4.3. E_Σ, as defined in 3.2, is a delicate EFSM type for activate and de-activate $\mathcal{R}_1$ and $\mathcal{R}_2$. It is assumed that the initial capability configuration, as defined in Section 1, has taken place.

The nature of the *service system adaptation manager* as well as the need and nature of a *policy composer* depends on the difference in income and penalty for the different QoS classes, as well as the cost for introducing a new server. If the income and penalty for premium service class is relatively higher than for an ordinary class, it can be profitable to disconnect some MP_O and let some MP_P get the service instead.

The set of *actions* $\mathcal{A}$ applied for the *service system adaptation manger* applied in Scenarios III and IV consists of *Disconnect-Client* ($\mathcal{A}_D$), *Decrease-Bit-Rate* ($\mathcal{A}_B$), *Initialize-Server* ($\mathcal{A}_I$) and *Remove-Server* ($\mathcal{A}_R$). $\mathcal{A}$ can be defined by (13) as follows:

$$\mathcal{A} \equiv \{ \mathcal{A}_D, \mathcal{A}_B, \mathcal{A}_I, \mathcal{A}_R \} \quad (13)$$

$\mathcal{A}_D$ tells MS to disconnect a list of suggested MP_O. $\mathcal{A}_B$ tells MS to reduce throughput of a list of suggested MP_P for a certain time period. $\mathcal{A}_I$ tells MS to initiate a new MS, while $\mathcal{A}_R$ will remove an MS.

6.2 RM specification

6.1.1 Service system adaptation manager

The reasoning condition set for the service system adaptation manager is defined as follows:

$$\Sigma \equiv \{ \Sigma_{T1}, \Sigma_{G1} \} \quad (14)$$

where the reasoning activation condition (Σ_{T1}) is $N_{Wait,P}+N_{Wait,O} > 0$ and the reasoning goal condition (Σ_{G1}) is $N_{Wait,P}+N_{Wait,O} = 0$. The messages sent and received between MS and the service system adaptation manager is defined by $msg(\Sigma_T, \mathcal{A}_i)$.

The rule set $\mathcal{X}$ for the service system adaptation manger in Scenario III and IV is defined as follows:

$$\mathcal{X} \equiv \{ \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3, \mathcal{X}_4 \} \quad (15)$$

$\mathcal{X}_1$ suggests $\mathcal{A}_D$ for disconnecting a list of suggested MP_O. $\mathcal{X}_2$ suggests $\mathcal{A}_B$ for reducing throughput of a list of suggested MP_P. $\mathcal{X}_3$ suggests $\mathcal{A}_I$ for initiating a new MS, while $\mathcal{X}_4$ suggests $\mathcal{A}_R$ for removing an MS. $\mathcal{X}_1$, $\mathcal{X}_2$, $\mathcal{X}_3$, $\mathcal{X}_4$ can be further defined by as follows:

$$\mathcal{X}_1 \equiv \; msg(\Sigma_{T1}, \mathcal{A}_i) \; \{ p_{Wait}(MP_O) < p_{Wait}(MP_P) \} \longrightarrow \mathcal{A}_i \doteq \mathcal{A}_D. \quad (16)$$

When $p_{Wait}(MP_O) < p_{Wait}(MP_P)$, $\mathcal{X}_1$ will be executed. The suggested action $\mathcal{A}_i$ will be instantiated ($\doteq$) as $\mathcal{A}_D$, The number of chosen MP_O will be calculated as:

$$\left(\frac{N_{Wait,P}\times X_{P,1Mbps}}{X_O}\right)$$

$$\mathcal{X}_2 \equiv \; msg(\Sigma_{T1}, \mathcal{A}_i) \; \{ p_{Wait}(MP_O) > m(MP_P, X_{P,1Mbps}) - m(MP_P, X_{P,800Kbps}) \} \longrightarrow \mathcal{A}_i \doteq \mathcal{A}_B. \quad (17)$$

When $p_{Wait}(MP_O) > m(MP_P, X_{P,1Mbps}) - m(MP_P, X_{P,800Kbps})$, $\mathcal{X}_2$ will be executed. The suggested action $\mathcal{A}_i$ will be instantiated ($\doteq$) as $\mathcal{A}_B$. The number of MP_P to decrease bandwidth is calculated from the bandwidth that the waiting MP_O is needed divided by the difference between the possible bit-rate required by MP_P as:

$$\left(\frac{N_{Wait,O} \times X_O}{X_{P,1Mbps} - X_{P,800Kbps}}\right)$$

$$\mathcal{X}_3 \equiv msg(\Sigma_{T1}, \mathcal{A}_i) \left\{ \frac{X_P \times N_{WaitP} + X_O \times N_{WaitO}}{C_{R,AL}} > 0.1 \right\} \longrightarrow \mathcal{A}_i \doteq \mathcal{A}_I. \quad (18)$$

As given in (18), when the ratio of throughput required by all waiting MP (as shown in the rule's condition) and the capacity of an MS access link is more than 0.1, $\mathcal{X}_3$ will be executed and $\mathcal{A}_I$ will be suggested. A new MS can be initialized in a node having sufficient capabilities, which are $C_{R,AL}$ as defined in (10).

$$\mathcal{X}_4 \equiv msg(\Sigma_{T1}, \mathcal{A}_i) \left\{ \frac{X_P \times N_{WaitP} + X_O \times N_{WaitO}}{C_{R,AL}} < 0.1 \right\} \longrightarrow \mathcal{A}_i \doteq \mathcal{A}_R. \quad (19)$$

$\mathcal{X}_4$ suggests $\mathcal{A}_R$ when additional MS are not needed based on the ratio of the throughput required by all waiting MP and the access link capacity. If ratio is less than 0.1, $\mathcal{A}_R$ will be suggested.

6.1.2 Policy composer

In Scenario IV, reasoning conditions of the policy composer are defined as follow:

$$\Sigma' \equiv \{ \Sigma_{T2} \doteq \Sigma_{T1}, \Sigma_{G2} \doteq \Sigma_{G1} \} \quad (20)$$

The policy composer will always be activated whenever the service system adaptation manager is activated and will be de-activated whenever the service system adaptation manager is de-activated.

Upon entering Σ_{T2}, the service system adaptation manager sends a message $msg(\Sigma_{T2}, \mathcal{A}_i)$ to the policy composer. The set of messages $\mathcal{Q}'$ sent and received between them is defined as follow:

$$\mathcal{Q}' \equiv \{ msg(\Sigma_{T2}, \mathcal{A}_i) \} \quad (21)$$

The set of *actions* $\mathcal{A}$ applied for the *policy composer* in the Scenario IV can be defined as follows:

$$\mathcal{A}' \equiv \{\mathcal{A}_G(\mathcal{X}_i), \mathcal{A}_T(\mathcal{X}_i) \} \quad (22)$$

$\mathcal{A}_G(\mathcal{X}_i)$ is an action for the calculation of the *accumulated goodness score* of a rule $\mathcal{X}_i$. $\mathcal{A}_T(\mathcal{X}_i)$ is an action to suspend $\mathcal{X}_I$ for a certain time period. The *goodness score of a rule* ($Qo\mathcal{X}_i$) during the monitoring time interval T is calculated by the percentage of the increased or decreased total income (m_T). The algorithm to calculate $Qo\mathcal{X}_i$ is as follows:

$$Qo\mathcal{X}_i = Qo\mathcal{X}_i + \frac{m_{T,t} - m_{T,t-1}}{m_{T,t}} \times 100 \quad (23)$$

where $m_{T,t}$ and $m_{T,t-1}$ are the total income during the current and previous monitoring interval respectively.

The rule set $\mathcal{X}'$ of the policy composer applied in Scenario IV is defined as follow:

$$\mathcal{X}' \equiv \{ \mathcal{X}'_1, \mathcal{X}'_2 \} \quad (24)$$

where $\mathcal{X}'_1$, $\mathcal{X}'_2$ (See Sec. 6.2.3) can be defined by (25) and (26) as follows:

$$\mathcal{X}'_1 \equiv msg(\Sigma_{T2}, \mathcal{A}_i) \{\mathcal{X}^{t-1} = \mathcal{X}_i \} \longrightarrow \mathcal{A}_i \doteq \mathcal{A}_G(\mathcal{X}_i). \quad (25)$$

When the policy composer finds that a rule $\mathcal{X}_i$ has been executed during the last interval t-1 ($\mathcal{X}^{t-1} = \mathcal{X}_i$), the policy composer executes $\mathcal{X}_1$. The suggested action $\mathcal{A}_i$ will be instantiated as $\mathcal{A}_G(\mathcal{X}_i)$.

$$\mathcal{X}_2 \equiv msg(\Sigma_{T2}, \mathcal{A}_i) \{ Qo\mathcal{X}_i < 0 \} \longrightarrow \mathcal{A}_i \doteq \mathcal{A}_T(\mathcal{X}_i). \quad (26)$$

When the goodness score of a rule is less than zero, the policy composer executes $\mathcal{X}_2$. The suggested action $\mathcal{A}_i$ will be instantiated as $\mathcal{A}_T(\mathcal{X}_i)$.

7. Results

The MP arrivals are modeled as a Poisson process with parameter λ_{QoS_Class}. The duration of streaming connections (d_{QoS_Class}) is constant. The quantity $\rho = ((\lambda_O \times d_O \times X_O)) + (\lambda_P \times d_P \times X_P)))/C_{I,AL}$ is the sum of traffic offered to MS access links. Note that ρ can be larger than the number of MS access links. The duration of streaming connections are set to 4 minutes, while the monitoring interval Δ is set to 1 minute.

Table 1– Income and penalty functions in *cost units*

	MP_O	MP_P X_I = 800Kbps	MP_P X_I = 1Mbps
m(QoS_Class, X_I) (per second)	$\frac{1}{60}$	$\frac{2}{60}$	$\frac{1.75}{60}$
p_{Wait}(QoS_Class) (per second)	$\frac{1}{3}$	$\frac{5}{3}$	$\frac{5}{3}$
p_{Dis}(QoS_Class) (per disconnection)	$\frac{5}{3}$	-	-

All Scenarios were tested for 500 minutes with two ρ values: 0.42 and 0.84. The MP_P arrival intensity is 15% of the total arrival intensity. The income and penalty functions in cost units are given in Table 1. The cost for using an extra MS is 417 cost units per Node per second.

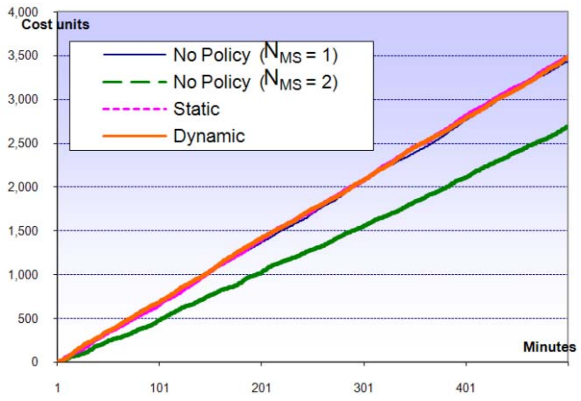


Figure 5 – the accumulated total income of all scenarios (ρ = 0.42)

Figure 5 illustrates the accumulated total income for all scenarios when ρ = 0.42. The values of accumulated total income of Scenario I (No Policy, N_{MS} = 1), Scenario III (static policies) and Scenario IV (dynamic polices) are identical, while the accumulated total income in the Scenario II (No Policy, N_{MS} = 2) is lower.

The low traffic implies that no rule is applied in Scenario III and IV. This made the outcome of Scenario I, III and IV identical. On the other hand, the cost of an extra server, which is not necessary for such arrival intensity, decreased the total income of the system.

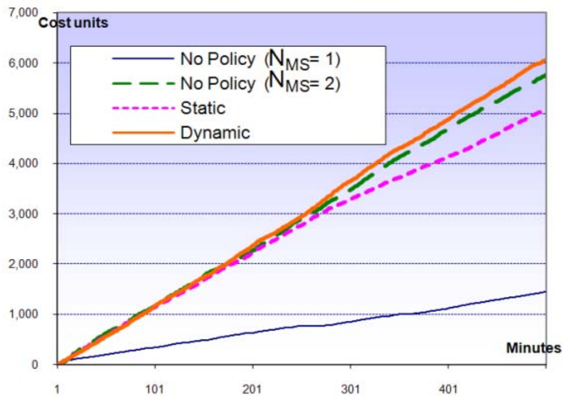


Figure 6 – the accumulated total income of all scenarios (ρ = 0.84)

Figure 6 Illustrates the accumulated total income of all scenarios when ρ = 0.84. As a result, the accumulated total income of Scenario I was much lower than the others as MP must wait to get the service.

Scenario II using two MS and no policy gives a very good result. There was no MP waiting during the test and no penalty was paid.

Scenario III and IV were started out with one MS. The number of MS were being increased or decreased by $\mathcal{X}_3$ and $\mathcal{X}_4$. In addition, $\mathcal{X}_1$ and $\mathcal{X}_2$ also manage the link capacity of MS by disconnecting MP_O or decrease the MP_P throughput. The accumulated income in Scenario IV was higher than for Scenario III.

It is observable on both Scenario III and IV that the use of $\mathcal{X}_4$, which will remove an MS, apparently reduced the system's accumulated total income. Having two servers all the time seems to be better for the high traffic, provided that the extra server cost is not too high. The dynamic policies suspend $\mathcal{X}_4$ for 50 minutes and thus lengthen the time where two MSs are in operation. In Scenario IV, $\mathcal{X}_4$ was executed 26 times comparing to 35 times in Scenario III.

For the present scenarios none of the non-policy scenarios (I and II) gave good results for both the low and high traffic case. The policy-based scenarios seem to be more suitable with respect to good results over a variety of system load conditions. The accumulated total income in Scenario III and IV also have the potential to be improved by changing the XML-based policies.

8. Conclusion

An architecture for policy-based adaptable service systems based on the combination of Reasoning Machines (RM) and Extended Finite State Machines (EFSMs) has been presented. The architecture comprises *service system framework, adaptation mechanisms* and *data model*. Policies have been introduced with the intension to increase flexibility in the adaptable service system specification and execution.

The service components constituting the *service system* are modeled by some combination of EFSM type and RM type. The RM, which is controlled by a specific purpose EFSM denoted as $EFSM_{\Sigma}$, is an independent component. The reasoning procedure applied by the RM is based on Equivalent Transform (ET).

The *Adaptation mechanism* uses policies to control service systems when it is entering a reasoning condition. The use of policy can be of two types: *static* or *dynamic*. In the static case the reasoning system *constituted by a service system adaptation manager* determines a list of suggested actions that will control the behavior of the service system. In the dynamic case an additional RM, denoted as the *policy composer,* is added. The *policy composer* is able to compose policy on-the-fly, and has the ability to estimate or evaluate the consequences of the rules of a policy based on their goodness scores.

The Data Model based on XML Equivalent Transformation (XET) is used to express system constraint, system status, reasoning conditions, rules and control rules. The XML-based specifications are readily executable by XET-based RM. This also represents a flexibility feature of the proposed architecture.

Four scenarios handling the capability configuration management of a music video on-demand service are presented with the intention to illustrate the use of the proposed architecture and the potential advantage of using dynamic policies. Scenario I and II use no policies. Scenario III uses static policies, while Scenario IV uses dynamic policies. There are situations where the use of no policy can be superior or equal to the use of policies. The selected system parameters can represent an optimal dimensioning. However, the same set of system parameters will likely not be optimal for other system traffic load cases. For the presented scenarios the use of no policy and one server is a good solution in the low traffic case, while the use of no policy and two servers is a good solution in the high traffic case.

In the given scenarios, the service system operated under static policies give a relatively high income in both low and high traffic. The service system operated under dynamic policies, however, has a performance which is superior or equal to other scenarios in both the low and the high traffic case. In addition to having the potential for providing optimal solutions covering dynamic traffic situations, the proposed architecture also is a flexible tool for the experimentation with alternative policies with respect to optimization.

References

[1] F. A. Aagesen, P. Supadulchai, C. Anutariya, and M. M. Shiaa, "Configuration Management for an Adaptable Service System," in *IFIP International Conference on Metropolitan Area Networks, Architecture, Protocols, Control, and Management*, Ho Chi Minh City, Viet Nam, 2005.

[2] D. Agrawal, K.-W. Lee, and J. Lobo, "Policy-Based Management of Networked Computing Systems," *IEEE Communications Magazine*, vol. 43, pp. 69-75, 2005.

[3] Y. Diao, J. L. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung, "A Control Theory Foundation for Self-Managing Computing Systems," *IEEE Journal on Selected Areas in Communications*, vol. 23, pp. 2213-2222, 2005.

[4] P. Supadulchai and F. A. Aagesen, "A Framework for Dynamic Service Composition," in *First International IEEE Workshop on Autonomic Communications and Computing (ACC 2005)*, Taormina, Italy, 2005.

[5] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-Based Self-Adaptation with Reusable Instrastructure," *Computer*, vol. 37, pp. 46-54, Oct 2004 2004.

[6] N. Samaan and A. Karmouch, "An Automated Policy-Based Management Framework for Differentiated Communication Systems," *IEEE Journal on Selected Areas in Communications*, vol. 23, pp. 2236-2247, 2005.

[7] R. Nasri, Z. Altman, and H. Dubreil, "Autonomic Mobile Network Management Techniques for Self-Parameterisation and Autoregulation," in *Smartnet 2006*, Paris, 2006.

[8] K. Akama, T. Shimitsu, and E. Miyamoto, "Solving Problems by Equivalent Transformation of Declarative Programs," *Journal of the Japanese Society of Artificial Intelligence*, vol. 13, pp. 944-952, 1998.

[9] S. Jiang and F. A. Aagesen, "XML-based Dynamic Service Behaviour Representation," in *NIK'2003*, Oslo, Norway, 2003.

[10] P. Supadulchai, "List of XML Variables, http://tapas.item.ntnu.no/wiki/index.php/XML_Variables," 2007.

Paper F

Towards Policy-Supported Adaptable Service Systems

Paramai Supadulchai, Finn Arve Aagesen and Patcharee Thongtra

In Proceedings of the *13th Eunice Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services*

University of Twente, the Netherlands, 2007.

Towards Policy-Supported Adaptable Service Systems

Paramai Supadulchai, Finn Arve Aagesen and Patcharee Thongtra

Department of Telematics
Norwegian University of Science and Technology (NTNU)
N7491 Trondheim, Norway
paramai@item.ntnu.no, *finnarve@item.ntnu.no*, *patt@item.ntnu.no*

Abstract. This paper presents a policy-supported architecture for adaptable service systems based on the combination of Reasoning Machines and Extended Finite State Machines. Policies are introduced to obtain flexibility with respect to specification and execution of adaptation mechanisms. The presented architecture covers two aspects: service system framework and adaptation mechanisms. The service system framework is a general framework for capability management. Adaptation mechanisms are needed for autonomous adaptation. The adaptation mechanisms can be based on static or dynamic policy systems. Capability management for of a simple music video-on demand service system with runtime simulation results based on the proposed architecture is presented.

1 Introduction

Networked service systems are considered. *Services* are realized by *service components*, which by their inter-working provide a service in the role of a *service provider* to a *service use*r. Service components are executed as software components in *nodes*, which are physical processing units such as servers, routers, switches and user terminals.

An adaptable service system is here defined as a service system which is able to adapt dynamically to changes in time and position related to users, nodes, capabilities, system performance, changed service requirements and policies. In this context, capability is defined as an inherent physical property of a node, which is used as a basis to implement services. Capabilities can be classified into resources, functions and data. Examples are CPU, memory, transmission capacity of connected transmission links, available special hardware, and available programs and data.

The software mechanisms used for implementing the functionality of the service components of adaptable service systems must be flexible and powerful. Service components based on the classical EFSM (Extended Finite State Machine) approach can be flexibly executed by using generic EFSM executing software components that are able to download and execute different EFSM-based specifications [1].

In addition to this type of flexibility the *EFSM-based* functionality can be supplemented by *reasoning-machine* (RM) based functionality, which makes policy-based specification and operation possible. *"Policies represent externalized logic that can determine the behavior of the managed systems"* [2]. In this paper *a policy is technically defined as a set of rules with related actions.* A *policy system* is a set of policies, and an *RM-based functionality is using a policy system to manage the behavior of a target system, which can be* another policy system. A *static policy system* has a non changeable set of rules and actions, while a *dynamic policy* system has a changeable set of rules and actions.

Policy-based software has a specification style, which is expressive and flexible. Software functionality based on policy-based specifications, however, also needs to be appropriately specified and validated. The validation aspect is outside the scope of this paper.

The main contribution of this paper is the presentation of a generic service framework for adaptable service systems that combines the use of EFSM-based and RM-based service components. In this context the reasoning machines can be used

a) as ordinary procedural services for EFSM-based service components
b) for instantiation and re-instantiation (i.e. after movement) of EFSM-based service components according to the availability and need of capabilities
c) to adapt the behavior of and capabilities allocated to instantiated EFSM-based service components in the nodes where they are instantiated

This paper has focus on issue c), but the framework presented can be used for a) and b) also. In general, adaptation needs appropriate mechanisms to guarantee the wanted results. For autonomous adaptation stable feedback loops [3], which control the performance, are needed. As the capabilities are limited, the access to the system must be controlled, and there must also be priority mechanisms that give priority to users which are willing to pay more and/or are in a higher need in situations with lack of capabilities.

The issues of policy-supported adaptable service system architecture are in this paper classified into 3 main aspects: A) Service system framework, B) Adaptation mechanism and C) Data model. *Service system framework* comprises abstraction, concepts and models. *Adaptation mechanism* concerns the use of the appropriate policies to control the service system when it is entering a state where RM functionality is needed. *Data model* concerns the data representation of the service system framework and adaptation mechanisms.

This paper comprises the aspects A) and B) only. For details about the data model, which is based on XML Equivalent Transformation language (*XET*), Common Information Model (CIM) and Resource Definition Framework (RDF), the reader is referred to [1] and [4]. The remaining part of this paper is structured as follows. Section 2 discusses related work. Section 3 presents the service system framework. Section 4 presents policy-based adaptation mechanism. Section 5 presents the models and results for example application cases related to capability management of a music video on-demand service. Section 6 gives summary and conclusions.

2 Related Work

Most of recent works related to policy-based adaptable service systems focus on the aspects A) and B) as defined in Section 1. Examples are [2, 5-10]. The aspect C) is supported by XML-based language in [2, 10], which is analogous to our used XML Equivalent Transformation (XET). However, [2] has a weak focus on the aspects A) and B), while [10] has a weak focus on A).

Considering the nature of the policies, [5] is preliminary aimed at static policies, while [6-10] are both using static and dynamic policies. Excluding [8], systems capable of dynamic policies [6-7, 9, 10] are based on proper feedbacks. The feedback loops in [5, 7, 9, 10] are used to evaluate the service system rather than policies. The loop in [6] evaluates policies.

However, the evaluation is based on complex mathematical equations and not by additional policy sets.

The adaptation mechanisms presented in this paper can use static as well as dynamic policies. Considering the dynamic policy, the rule-based modification of the policy managing the service system can be composed at run-time.

The use of dynamic policies in [9, 10] as well as in this paper also aims at being a flexible tool for the experimentation with alternative policies with respect to optimization.

3 Service System Framework

The concept *capability* was defined in Section 1. *Capability performance measures* are the concepts used for the performance modeling, dimensioning, analyzing, monitoring and management of capabilities. Capability performance measures comprise capability *capacity*, capability *state* and capability *Quality of Service* (QoS) measures (e.g. traffic and availability measures). *Service performance measures* are performance measures related to the service provided to the service user (e.g. QoS measures) as well as service system state measures.

An executing service system consists of executing service components which are *instances* of *service component type*s. The functionality types are *EFSM types* and *RM types.* The basic functionality of the service components, however, are based on EFSMs supported and/or controlled by RMs. EFSM components will have requirements with respect to *capability* and *service performance* to be able to perform their intended functionality (Fig. 1). These requirements are denoted as required capability and service performance. The capability and service performance of an executing service system are denoted as inherent capability and service performance.

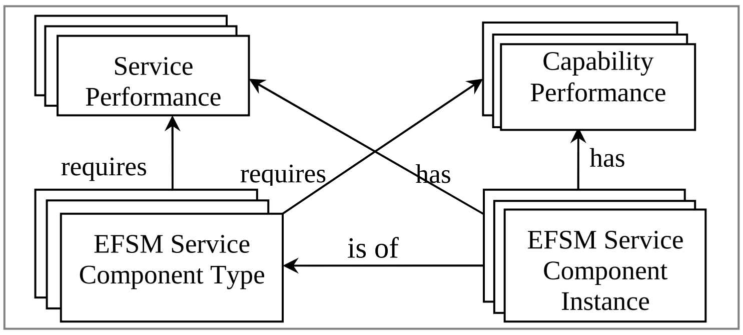


Fig. 1. EFSM part of Service System – Concept Structure

Capability management (CM) is an important function within an adaptable service system and comprises: 1) service system capability initialization, 2) capability allocation adaptation and 3) capability re-initialization. *Service system capability initialization* is the allocation of the capabilities for the service components to be distributed and instantiated. Capabilities are allocated according to the system performance requirements of the EFSM components of a service system. *Capability allocation adaptation* is the monitoring of the performance of the executing service system and the reallocation of capabilities within the executing service systems. In situations when the instantiated service systems are unable to adapt satisfactory, capability management can initiate a *service system capability re-initialization* for a re-distribution and re-instantiation of the service system.

As a basis for the optimal adaptation, *service level agreements* (SLA) are needed between the *service users* and the *service provider*. The service provider view of this service level agreement can in this context be considered as a part of executing service components. A number of QoS levels can exist. The agreement can contain elements such as: agreed QoS levels, required capabilities, required system performance, payment for the service in case of agreed QoS level and payments for the service in case of reduced QoS level. A service level agreement class (SLA class) defines provided service user functionalities as well as agreed QoS parameter and cost values for a group of service users with different degree of satisfactions and cost.

In the following a formalized service framework model is presented. The following concepts are defined:

	Functionality set of an EFSM type
	Functionality set of an EFSM instance
	Functionality set of a RM type
	Functionality set of a RM instance
C	Capability performance measures set
$\hat{C}R$	Required capability performance set for an EFSM-based service component type
$\hat{C}I$	Inherent capability performance set of an executing EFSM-based service component
$\hat{C}A$	Set of available capabilities in nodes
S	Service performance measures set
$\hat{S}R$	Required service performance set for an EFSM-based service component type
$\hat{S}I$	Inherent service performance set of an executing EFSM-based service component
I	Income functions set for the service components constituting a service. These functions will depend on the system performance.

The EFSM type E and the RM type $\mathcal{R}$ are defined (≡) as follows:

$$E \equiv \{ S_M, S_I, V, P, M(P), O(P), F_S, F_O, F_V \} \quad \textbf{(1)}$$

$$\mathcal{R} \equiv \{ \mathcal{Q}, \mathcal{F}, \mathcal{P}, \mathcal{T}, \mathcal{E}, \Sigma \} \quad \textbf{(2a)}$$

$$\mathcal{P} \equiv \{ \mathcal{X}, \mathcal{A} \} \quad \textbf{(2b)}$$

Concerning E, S_M is the set of states, S_I is the initial state, V is a set of variables, P is a set of parameters, M(P) is a set of input signal with parameters, O(P) is a set of output signal with parameters, F_S is the state transition function ($F_S = S \times M(P) \times V$), F_O is the output function, ($F_O = S \times M(P) \times V$) and F_V are the functions and tasks performed during a specific state transition such as computation on local data, communication initialization, database access, etc.

Concerning $\mathcal{R}$ and $\mathcal{P}$, $\mathcal{Q}$ is the set of messages, $\mathcal{F}$ is a generic *reasoning procedure*, $\mathcal{P}$ is a policy system which consists of a set of rules $\mathcal{X}$ and a set of actions $\mathcal{A}$, $\mathcal{T}$ is a set of *system* *constraints* and $\mathcal{E}$ is a set of *performance data*. The *reasoning procedure* is the procedure applied by RM to select the appropriate actions. The performance data represents the inherent performance of the targeted system. The system constraints represent the variables of the system and the defined constraints and relationships between variables. The policy rules are based on the variables of the constraints. Σ is a set of reasoning conditions defined by *trigger conditions* Σ_T, *and goal conditions* Σ_G. RM functionality is activated when a Σ_T is detected until a Σ_G is reached. When a trigger condition is true, the reasoning procedure transforms $\mathcal{Q}_i$ to $\mathcal{Q}_j$ by using $\mathcal{P}$ to match the system constraints $\mathcal{T}$ against the *performance data* $\mathcal{E}$ and a set of suggest actions $\{\mathcal{A}_i, \mathcal{A}_j, \mathcal{A}_k \ldots\} \subseteq \mathcal{A}$. These actions may also set the next state and values of the variables of EFSM-based service component instances. The *reasoning procedure* is based on *Equivalent Transformation* (*ET*) [11], which solves a given problem by transforming it through repetitive application of (semantically) equivalent transformation rules.

The RM functionality will need EFSM support for the continuous updating of $\mathcal{T}$, $\mathcal{E}$ and Σ, and for the activation and deactivation of the reasoning machines. This is done by EFSMs, and in this case $\mathcal{T}$, $\mathcal{E}$ and Σ are considered as common data for the EFSMs and the associated RM-based functionality. A dedicated EFSM E_Σ has the duty to inspect the reasoning condition and to activate and to deactivate the reasoning machine.

4 Policy-Based Adaptation Mechanism

4.1 System constraints, performance data and reasoning conditions

The elements $\mathcal{T}$ and $\mathcal{E}$ of an RM as defined in Section 3 depend on the structuring and the nature of the reasoning functionality. A *reasoning cluster, which* is an independent unit with respect to reasoning, is a collection of EFSM-based service components with an associated *reasoning system* constituted by one or more *reasoning machine*s. A reasoning cluster has a set of associated income functions I. The elements $\mathcal{T}$ and $\mathcal{E}$ of a reasoning cluster with available capabilities from N_{Node} nodes, consisting of K EFSM-based service component types and L_k instances of an EFSM-based service type k are defined as follows:

$$\mathcal{T} \quad \equiv \quad Expr\,\{S, C, I, (E_k, \hat{S}_{R,k}, \hat{C}_{R,k};\, k = [1, K])\} \tag{3}$$

$$\mathcal{E} \quad \equiv \quad \{((\hat{E}_{lk}, \hat{S}_{I,lk}, \hat{C}_{I,lk};\, l = [1, L_k]),\, k = [1, K]),\ (\hat{C}_{A,n};\, n = [1, N_{Node}])\} \tag{4}$$

The function $Expr\{X_i;\ i = [1, I]\}$ in (3) symbolizes the set $\{X_i;\ i = [1, I]\}$ and also some set of logical functions based on the elements of the set. The system constraints $\mathcal{T}$ related to a reasoning cluster comprise the EFSM functionality sets of the EFSM-based service component types, required capability and service performance, as well as the income functions for the reasoning cluster. The performance data $\mathcal{E}$ defined in (4) is a set of the inherent capability and service performance for all instances of EFSM-based service components in the reasoning cluster, as well as available capabilities of the nodes that potentially can contribute their capabilities for the EFSM-based functionality of the reasoning cluster.

The components constituting the *reasoning condition* Σ are the states and variables of the EFSM-based service component types, and the capability and service performance measures C and S as given in (5).

$$\Sigma \quad \equiv \quad Expr\,\{S, C, (E_k, \hat{S}_{R,k}, \hat{C}_{R,k};\, k = [1, K])\} \tag{5}$$

Capability Management (CM) as defined in Section 3 goes beyond the boundaries of an individual reasoning cluster as well as an individual service system. This means that CM in general must be handled by a common distributed algorithm or by a centralized reasoning cluster.

4.2 Policy-based adaptation using static policies

The adaptation mechanism using static policies is illustrated in Fig. 2. The rules $\mathcal{X}$ are unchangeable. When the service systems enter a Σ_T, Service System Adaptation Manager ($\mathcal{R}_1$) is activated and tries to lead the system back to a goal state Σ_G. $\mathcal{R}_1$ is de-activated when service systems enter Σ_G.

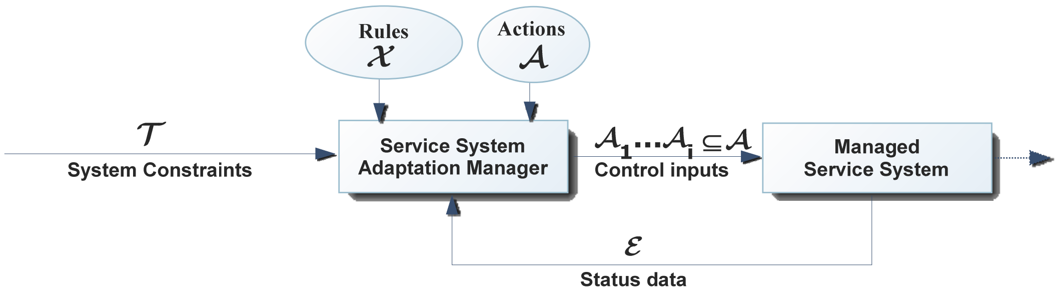
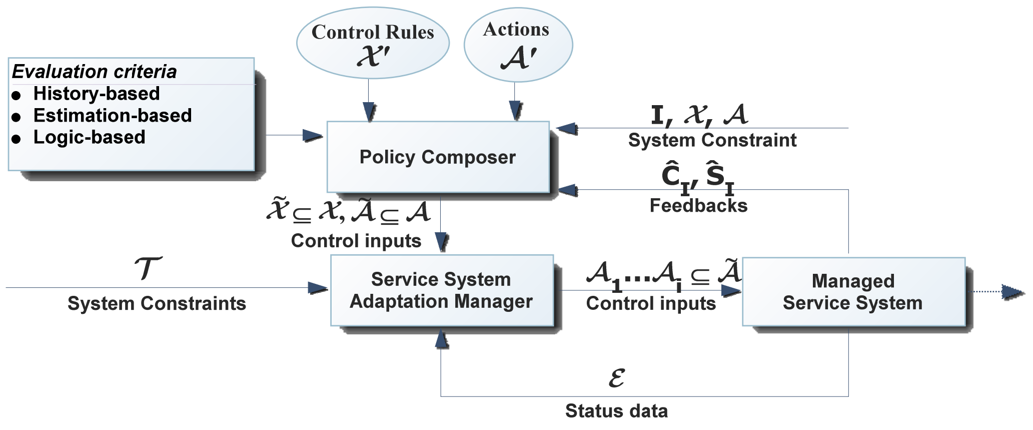


Fig. 2. Policy-based adaptation using static policies

4.3 Policy-based adaptation using dynamic policies



Evaluation criteria
● History-based
● Estimation-based
● Logic-based
Control Rules $\mathcal{X}'$
Actions $\mathcal{A}'$
Policy Composer
$\mathbf{I}, \mathcal{X}, \mathcal{A}$
System Constraint
$\hat{\mathbf{C}}_I, \hat{\mathbf{S}}_I$
Feedbacks
$\tilde{\mathcal{X}} \subseteq \mathcal{X}, \tilde{\mathcal{A}} \subseteq \mathcal{A}$
Control inputs
$\mathcal{T}$
System Constraints
Service System Adaptation Manager
$\mathcal{A}_1 \dots \mathcal{A}_i \subseteq \tilde{\mathcal{A}}$
Control inputs
Managed Service System
$\mathcal{E}$
Status data

Fig. 3. Policy based adaptation based on dynamic policies

The adaptation mechanism using dynamic policies is illustrated in Fig. 3. In addition to the Service System Adaptation Manager ($\mathcal{R}_1$) a Policy Evaluator ($\mathcal{R}_2$) is used.

A generic rule-based reasoning system with dynamic policy can be defined by (6a, 6b, 6c and 6d) as follows:

$$\mathcal{R}_1 \equiv \{ \mathcal{Q}, \mathcal{F}, \tilde{\mathcal{P}}, \mathcal{T}, \mathcal{E}, \Sigma \} \quad \textbf{(6a)}$$

$$\tilde{\mathcal{P}} \equiv \{ \tilde{\mathcal{X}}, \tilde{\mathcal{A}} \} \quad \textbf{(6b)}$$

$$\mathcal{R}_2 \equiv \{\, \mathcal{Q}', \mathcal{F}, \mathcal{P}', \mathcal{T}', \mathcal{E}', \Sigma' \,\} \qquad \textbf{(6c)}$$

$$\mathcal{P}' \equiv \{\, \mathcal{X}', \mathcal{A}' \,\} \qquad \textbf{(6d)}$$

where $\mathcal{T}' = \{\mathrm{I}, \mathcal{X}, \mathcal{A}\}$ and $\mathcal{E}' = \{\hat{\mathrm{C}}_{\mathrm{I}}, \hat{\mathrm{S}}_{\mathrm{I}}\}$. $\mathcal{Q}'$ is a set of messages between $\hat{\mathcal{R}}_1$ and $\hat{\mathcal{R}}_2$. $\mathcal{X}'$ is a set of control rules that can re-order the priority of the rules, activate and de-activate the rules and change rules' constraints. The policy evaluator evaluates the system policy at runtime based on *evaluation criteria*, *reference inputs* and *feedbacks*. Income functions are used as reference inputs, while the feedbacks are system performance measures. Evaluation criteria can in general be history-based and prediction-based. This paper is only using history-based evaluation, which determines the consequences of the rules in the past using service performance measures. The prediction-based evaluation determines the consequences of rules in the future based on mathematical equations represented by $\mathcal{X}'$.

Dynamic policies need a certain period to evaluate the consequences of the rules used. A measure for *the learning ability* is the *learning time* ($\mathrm{T_L}$), which is the time needed by the system to properly evaluate the rules. The *learning time* $\mathrm{T_L}$ depends on the service performance measures used by the evaluation algorithm. However, there is no unique and easy way to define $\mathrm{T_L}$.

5 Application Examples

5.1 The application cases

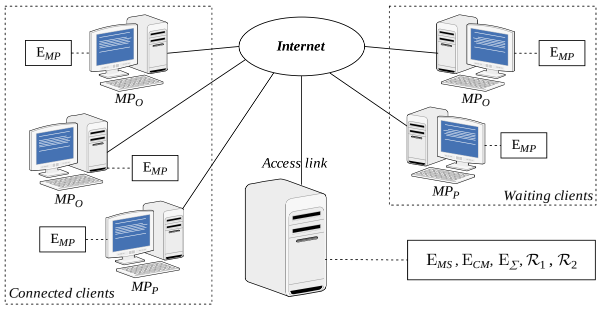


Fig. 4. A music video on-demand service system; $\mathrm{E_{MS}}$: Media server type, $\mathrm{E_{MP}}$: Media player type, $\mathrm{E_{CM}}$: Capability manager type, E_Σ: Dedicated EFSM type for controlling the reasoning mechanism, $\mathcal{R}_1$: Service system adaptation manager type, $\mathcal{R}_2$: Policy evaluator type.

Five application cases (Case I-V) for a simple service system handling the capability management for a music video on-demand service is presented. The intention is to illustrate the use of the proposed policy-based service system architecture, and the potential advantages of using dynamic policies. The Cases I - III use no policy, Case IV uses static policies, while Case V uses dynamic policies. The service system is constituted by one or more media servers (MS) streaming media files to media players (MP) (Fig. 4). The numbers of MS used in Case I, II and III are fixed (one, two and three respectively), while the number in Case IV and V can vary from one to three.

The basic EFSM types constituting the capability management system are media server handler ($\mathrm{E_{MS}}$), media player handler ($\mathrm{E_{MP}}$) and capability manager ($\mathrm{E_{CM}}$)

The capability manager, which operation is based on policy based adaptation, is used in Case IV and V. According to the definition of capability management in Section 3, service system capability initialization and re-initialization is not included in the example. This means that only capability allocation adaptation is considered. With reference to the concepts service system adaptation manger and policy evaluator as defined in Section 4, the capability manager now has the role of a service system adaptation manager, and the policy evaluator is the system determining the policies to be used of the capability manager.

In the fixed policy case (Case IV) E_{CM} is supported by a rule-based reasoning system $\mathcal{R}_1$, and in the dynamic policy case (Case V) E_{CM} is supported by $\mathcal{R}_1$ and $\mathcal{R}_2$. The EFSM type E_{Σ} is the dedicated EFSM that inspects the reasoning conditions Σ and activates/deactivates the reasoning mechanisms.

The MS's required access link capacity $C_{R,AL}$ is set to 100 Mbps. The number of MPs that can use the service is limited by the MS access link capacity. An MP belongs to a *SLA_Class*. In the example two classes are applied: premium (MP_P) and ordinary (MP_O). Three different streaming throughput bit-rates (X) are offered, 500Kbps, 600 Kbps and 1Mbps. MP_O connections are 500Kbps (X_O) while MP_P connections can be either 600Kbps or 1Mbps (X_P).

The service level agreements comprise *required streaming throughput*, *maximum waiting time, payment for the service* and *penalties for not satisfying the service*. The required streaming throughput of MP_O and MP_P are X_O and $X_{P,}$ respectively.

The mechanisms used by the capability manager are to let client wait, to disconnect ordinary clients, to decrease the throughput of the premium clients and to change the number of media servers.

When the required streaming throughput cannot be provided, an MP may have to wait until some connected MPs have finished using the service. This will result in money payback to the waiting MPs. An MP_O can be disconnected, while an MP_P may have to reduce the throughput. If a client is disconnected, the service provider pays a penalty. The maximum waiting time for MP_P and MP_O are 60 seconds and infinite respectively.

The service performance measures $\hat{S}_I$ are the number of connected and waiting premium and ordinary clients ($N_{Con,P}$, $N_{Con,O}$, $N_{Wait,P}$, $N_{Wait,O}$), the number of disconnected MP_O ($N_{Dis,O}$), the number of MS (N_{MS}), inherent streaming throughput (X_I), the number of available nodes (N_{Node}) and the accumulated service time and waiting time of premium and ordinary clients ($T_{Serv,P}$, $T_{Serv,O}$, $T_{Wait,P}$, $T_{Wait,O}$). These values are observed per monitoring interval Δ.

A *unit* is the price paid by an ordinary customer for *one second streaming* of the rate 500 Kbps. The income function for the service provider is $m(SLA\_Class, X_I)$ (units/s). The penalty function for waiting is $p_{Wait}(SLA\_Class)$ (units/s). The penalty function for disconnections is $p_{Dis}(SLA\_Class)$ (units/disconnection). The cost function for adding a new server is p_{Ser} (units/s per Node). The total income function (m_T) during the monitoring interval Δ is:

$$\begin{aligned} m_T = \; & m(MP_O, X_{I,O})\times T_{Serv,O} + m(MP_P, X_{I,P})\times T_{Serv,P} - p_{Wait}(MP_O)\times T_{Wait,O} \\ & - p_{Wait}(MP_P)\times T_{Wait,P} - p_{Dis}(MP_O)\times N_{Dis,O} - p_{Ser}\times(N_{MS}-)\times\Delta \end{aligned} \quad (7)$$

The reasoning machine supported capability manager will try to maximize the total income. The service system is realized as one reasoning cluster as illustrated in Fig. 4. The nature of the *service system adaptation manager* as well as the need and nature of a *policy evaluator* depends on the difference in income and penalty for the different SLA classes, as well as the cost for introducing a new server. If the income and penalty for premium service class is relatively higher than for an ordinary class, it can be profitable to disconnect some MP_O and let some MP_P get the service instead.

The specification of the behavior of the *service system adaptation manager* used for the Cases IV and V, and the *policy evaluator* applied for the Case V is given in Appendix.

5.2 Results

Table 1. Income and penalty functions

	MP_O	MP_P (X_I = 600Kbps)	MP_P (X_I = 1Mbps)
m(SLA_Class, X_I) / s	1	1.875	2
p_{Wait}(SLA_Class) / s	5	10	10
p_{Dis}(SLA_Class) / disconnection	10	-	-

The MP arrivals are modeled as a Poisson process with parameter λ_{SLA_Class}. The duration of streaming connections d_{SLA_Class} is constant. The quantity $\rho = ((\lambda_O \times d_O \times X_O) + (\lambda_P \times d_P \times X_P))/C_{I,AL}$ is the traffic per an MS access link. Intuitively, the system with $\rho \leq 1$ needs at least one server while the system with $1 \leq \rho \leq 2$ needs at least two servers and so on. The MP_P arrival intensity is 15% of the total arrival intensity. The duration of streaming connections are set to 10 minutes, while the monitoring interval Δ is set to 1 minute. MPs stop waiting after 10 minutes. The income and penalty functions in units are given in Table 1.The cost for using an extra MS is 833 units/s per Node.

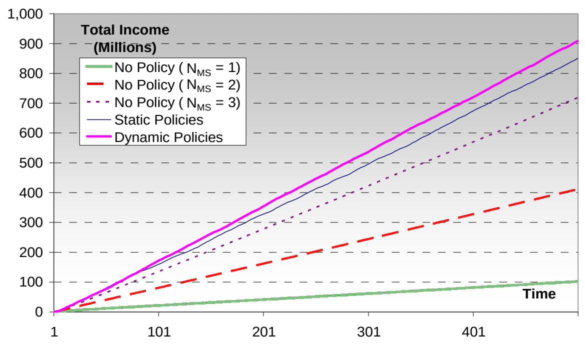


Fig. 5. Accumulated total income for ρ = 3.45

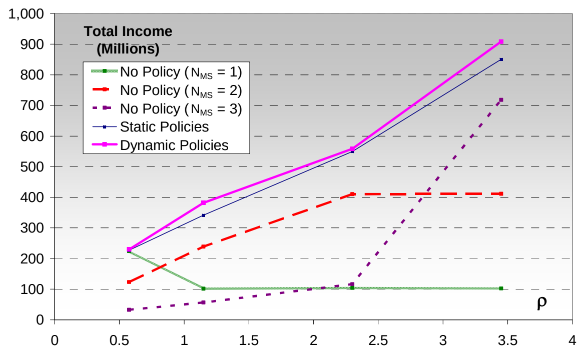
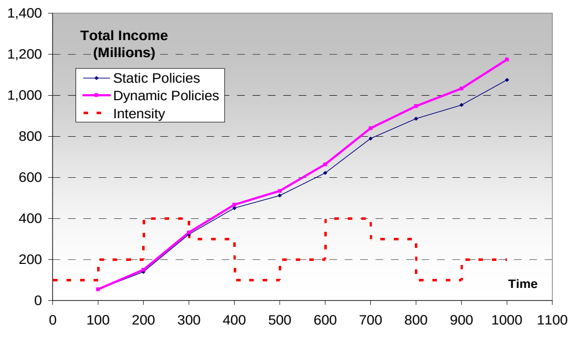


Fig. 6. Accumulated income at 500th ms

Fig. 5 illustrates the accumulated total income when ρ = 3.45. The value 3.45 is chosen to compare the no-policy scenarios with N_{MS} = 1, 2, or 3 and as well as the static and dynamic policy scenarios. The accumulated total incomes of cases with no policy are relatively lower than those with policies.

Fig. 6 illustrates the values of accumulated total income at the 500th minute for the ρ values: 0.56, 1,2, 2.3 and 3.5. The systems with no policy produce good results with a certain load region. The systems operated under policies produced higher accumulated total income independent of load region. Dynamic policies give relatively better result. These cases also have the potential improvement by changing the policies.

Fig. 7 shows the system behavior for Case IV and V when the traffic is being increased or decreased (the value of ρ varies as a function of time). The time with ρ at a fixed level is denoted as the ρ *period*. The dotted line shows the variation of ρ, which can take the values 0.5, 1, 1.5 and 2 times of ρ = 1.44. The ρ period, which is $10\times d_{\text{SLA\_Class}}$, provides much time for the system for learning the consequences of the rules being applied. Case V gives a better result.



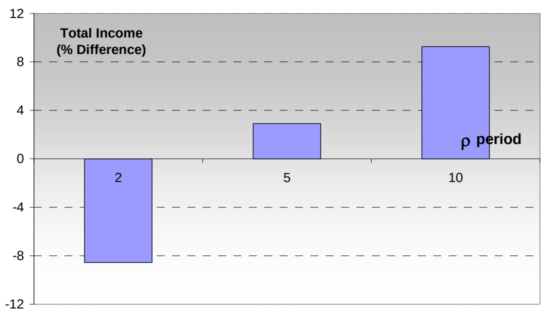


Fig. 7. Accumulated total income

Fig. 8. Comparison of Case IV&V

Fig. 8 shows a comparison between Case IV and V for different ρ periods. The figure shows the difference between the values of accumulated total income after 500 minutes. When the ρ period is small, Case IV may give better result because the system need more learning time (T_L). The T_L value falls between $2\times$ and $5\times d_{\text{SLA\_Class}}$.

The use of $\mathcal{X}_3$, $\mathcal{X}_4$ (see Appendix), which will add or remove an MS, affects the system's accumulated total income. Having more MS all the time is better for high traffic while having few MS all the time is better for low traffic. The policy evaluator learned this by observing the consequences of $\mathcal{X}_3$ and $\mathcal{X}_4$. The ability to learn can also be improved by appropriately selecting service performance measures and algorithms.

6 Conclusion

An architecture for policy-based adaptable service systems, based on the combination of Reasoning Machines (RMs) and Extended Finite State Machines (EFSMs) has been presented. Policies have been introduced with the intension to increase flexibility in the system specification and execution.

The adaptation mechanism uses policies to control service systems when it is entering a reasoning condition. The use of policy can be of two types: *static* or *dynamic*. In the static case the reasoning system *constituted by a service system adaptation manager* determines a list of suggested actions that will control the behavior of the service system. In the dynamic case an additional RM, denoted as the *policy evaluator*, is added. The *policy evaluator* is able to compose policy on-the-fly, and has the ability to estimate or evaluate the consequences of the rules of a policy based on their accumulated goodness scores.

Five application cases handling the capability management of a music video on-demand service are presented. The intention is to illustrate the use of the proposed architecture and demonstrate the potential advantage of using dynamic policies. Case I, II and III use no policies. Case IV uses static policies, while Case V uses dynamic policies. Only capability allocation adaptation is considered. There are situations where the use of no policy can be superior or equal to the use of policies. The selected system parameters can represent an optimal dimensioning. However, the same set of system parameters will likely not be optimal for other system traffic load cases. The service system operated under static policies give a relatively high income in both low and high traffic. The service system operated under dynamic policies, however, has a performance which is superior or equal to other application cases. Nevertheless, the service system operated under dynamic policies needs a certain period of time denoted as *learning time* to learn the consequences of policies in order to provide superior performance.

The proposed architecture is also a flexible tool for the experimentation with alternative policies with respect to optimization.

References

1. F. A. Aagesen, P. Supadulchai, C. Anutariya, and M. M. Shiaa, "Configuration Management for an Adaptable Service System," in IFIP International Conference on Metropolitan Area Networks, Architecture, Protocols, Control, and Management, Ho Chi Minh City, Viet Nam, 2005.
2. D. Agrawal, K.-W. Lee, and J. Lobo, "Policy-Based Management of Networked Computing Systems," *IEEE Communications Magazine,* vol. 43, pp. 69-75, 2005.
3. Y. Diao, J. L. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung, "A Control Theory Foundation for Self-Managing Computing Systems," *IEEE Journal on Selected Areas in Communications,* vol. 23, pp. 2213-2222, 2005.
4. P. Supadulchai and F. A. Aagesen, "A Framework for Dynamic Service Composition," in *First International IEEE Workshop on Autonomic Communications and Computing (ACC 2005)*, Taormina, Italy, 2005.
5. D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-Based Self-Adaptation with Reusable Instrastructure," *Computer,* vol. 37, pp. 46-54, Oct 2004 2004.
6. N. Samaan and A. Karmouch, "An Automated Policy-Based Management Framework for Differentiated Communication Systems," *IEEE Journal on Selected Areas in Communications,* vol. 23, pp. 2236-2247, 2005.
7. R. Nasri, Z. Altman, and H. Dubreil, "Autonomic Mobile Network Management Techniques for Self-Parameterisation and Auto-regulation," in *Smartnet 2006,* Paris, 2006.
8. Y. Kanada, "Dynamically Extensible Policy Server and Agent," in *Proceedings of the 3rd Int'l Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, 2002.
9. H. Chan and T. Kwok, "A Policy-based Management System with Automatic Policy Selection and Creation Capabilities using a Singular Value Decomposition Technique," in *Proceedings of the 7th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06),* 2006.
10. R. J. Anthony, "A Policy-Definition Language and Prototype Implementation Library for Policy-based Autonomic Systems," in *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on,* 2006.

11. K. Akama, T. Shimitsu, and E. Miyamoto, "Solving Problems by Equivalent Transformation of Declarative Programs," *Journal of the Japanese Society of Artificial Intelligence,* vol. 13, pp. 944-952, 1998.

Appendix. Reasoning Machine Specifications

1. Service system adaptation manager (Case IV and V)

The set of actions $\mathcal{A}$ applied by the service system adaptation manger is:

$$\mathcal{A} \equiv \{ \mathcal{A}_D, \mathcal{A}_B, \mathcal{A}_I, \mathcal{A}_R \} \qquad \textbf{(A.1)}$$

$\mathcal{A}_D$ *(Disconnect-Client)* tells MS to disconnect suggested MP_O. $\mathcal{A}_B$ *(Decrease-Bit-Rate)* tells MS to reduce throughput of suggested MP_P for a certain time period. $\mathcal{A}_I$ *(Initialize-Server)* tells MS to initiate a new MS, while $\mathcal{A}_R$ *(Remove-Server)* will remove a MS. Concerning the reasoning condition set $\Sigma \equiv \{ \Sigma_{T1}, \Sigma_{G1} \}$, the reasoning activation condition Σ_{T1} is $N_{Wait,P}+N_{Wait,O} > 0$ and the reasoning goal condition Σ_{G1} is $N_{Wait,P}+N_{Wait,O} = 0$. The rule set $\mathcal{X}$ for the service system adaptation manger is:

$$\mathcal{X} \equiv \{ \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3, \mathcal{X}_4 \} \qquad \textbf{(A.2)}$$

$\mathcal{X}_1$ suggests $\mathcal{A}_D$ for disconnecting a list of suggested MP_O when $p_{Wait}(MP_O) < p_{Wait}(MP_P)$. The number of MP_O is calculated from $N_{Wait,P} \times X_{P,1Mbps} / X_O$. $\mathcal{X}_2$ suggests $\mathcal{A}_B$ for reducing throughput of a list of suggested MP_P when $p_{Wait}(MP_O) > m(MP_P, X_{P,1Mbps}) - m(MP_P, X_{P,600Kbps})$. The number of MP_P to decrease bandwidth is calculated from $N_{Wait,O} \times X_O / (X_{P,1Mbps} - X_{P,600Kbps})$. $\mathcal{X}_3$ suggests $\mathcal{A}_I$ for initiating a new MS when $X_P \times N_{Wait,P} + X_O \times N_{Wait,O} / C_{R,AL} > 0.1$. $\mathcal{X}_4$ suggests $\mathcal{A}_R$ for removing an MS when $X_P \times N_{Wait,P} + X_O \times N_{Wait,O} / C_{R,AL} < 0.1$.

2. Policy evaluator (Case V)

The policy evaluator will be activated and de-activated whenever the service system adaptation manager is activated and de-activated. So we have activation condition $\Sigma_{T2} \doteq \Sigma_{T1}$ (means '*is instantiated as*'), and goal condition Σ_{G2} Σ_{G1}. The set of actions $\mathcal{A}$ applied by the the policy evaluator is:

$$\mathcal{A}' \equiv \{ \mathcal{A}_G(\mathcal{X}_i), \mathcal{A}_T(\mathcal{X}_i) \} \qquad \textbf{(A.3)}$$

$\mathcal{A}_G(\mathcal{X}_i)$ is an action for the calculate of the *accumulated goodness score* of a rule $\mathcal{X}_i$. $\mathcal{A}_T(\mathcal{X}_i)$ is an action to suspend $\mathcal{X}_I$ for a certain time period. The *goodness score of a rule* ($Qo\mathcal{X}_i$) during the monitoring time interval T is calculated by the percentage of the increased or decreased total income (m_T). The algorithm to calculate $Qo\mathcal{X}_i$ is as follows:

$$Qo\mathcal{X}_I = Qo\mathcal{X}_i + \frac{m_{T,t} - m_{T,t-1}}{m_{T,t}} \times 100 \qquad \textbf{(A.4)}$$

where $m_{T,t}$ and $m_{T,t-1}$ are the total income during the current and previous monitoring interval respectively. The rule set $\mathcal{X}'$ of the policy evaluator is:

$$\mathcal{X}' \equiv \{ \mathcal{X}'_1, \mathcal{X}'_2 \} \qquad \textbf{(A.5)}$$

$\mathcal{X}'_1$ calculate the goodness score of the rule used during the last interval using the action $\mathcal{A}_G(\mathcal{X}_i)$, and $\mathcal{X}'_2$ suspends rules using the action $\mathcal{A}_T(\mathcal{X}_i)$ when their goodness scores are below zero.

Paper G

A Capability-based Service Framework for Adaptable Service Systems

Finn Arve Aagesen and Paramai Supadulchai

In Proceedings of *The 2nd International Conference on Advances in Information Technology* (IAIT2007)

Bangkok, Thailand, November 1-2, 2007

A Capability-based Service Framework for Adaptable Service Systems

Finn Arve Aagesen and Paramai Supadulchai
Institute of Telematics, NTNU
N7491 Trondheim, Norway
finnarve@item.ntnu.no, paramai@item.ntnu.no

Abstract

This paper presents a capability-based service framework for adaptable service systems. The paper has focus on: I) Adaptability properties, II) The architecture solution needed to meet these properties, and III) Capability configuration management functionality.

Adaptability is based on flexibility. Accordingly, basic rearrangement flexibility must be provided. But the framework must in addition have necessary concepts, features and functionality that make it possible to adapt to various traffic situations and failure states.

The architecture solution has a computing and a service functionality dimension. The computing architecture is based on a theatre metaphor, where actor, manuscript, role figure and capability are core concepts. Fundamental QoS concepts are capability, capability performance, service performance and service level agreements. Actors, which are the basis for the implementation of the service functionality, can be Extended Finite State Machines or Reasoning Machines. Actors are able to download and execute EFSM- and RM-based manuscripts.

Models for capability configuration management functionality are presented. Reasoning machines are used for capability configuration management in general as well as for policy adaptation.

1. Introduction

Networked service systems are considered. Services are realized by service components, which by their inter-working provide a service in the role of a service provider to a service user. Service components are executed as software components in nodes, which are physical processing units such as servers, routers, switches, PCs and mobile phones. A *service framework* is here defined as the overall structural and behavior framework for the specification and execution of services. Networked services have during more than two decade been is an important research topic. Topics include Intelligent Networks and TINA (Tele-communication Information Networking Architecture) [1], Mobile Agents and Active and Programmable Networks [2] in the 80-ies and 90-ies. Focus was on flexibility and efficiency in the definition, deployment and execution. This focus has now been changed into *adaptability* and *evolution*. We have entered an era with a high degree of flexibility. To utilize the flexibility potential as a foundation for adaptability, the attributes of services and nodes must be appropriately formalized, stored and made available.

An adaptable service system is here defined as a service system which is able to adapt dynamically to changes in time and position related to users, nodes, capabilities, system performance, changed service requirements and policies. In this context, capability is defined as an inherent physical property of a node, which is used as a basis for implementing services (See Section 2). This definition of adaptability covers a wide spectrum of functionality, and does also include autonomic communication [3, 4].

This paper presents some issues related to service frameworks for adaptability. The issues presented are part of *TAPAS* (TAPAS = Telematics Architecture for Play-based Adaptable Systems. See [5] and the URL: http://tapas.item.ntnu.no. The paper has focus on three issues: I) Adaptability properties, II) Architecture solution and III) Capability configuration management functionality

The adaptability properties presented in Section 2 are the basis for the architecture solution which is presented in Section 3-4. The architecture has a computing dimension and a service functionality dimension. The computing architecture, which is based on a theatre metaphor, is presented in Section 5. An actor playing a role is either an Extended Finite State Machines (EFSM) or a Reasoning Machines (RM).

Issue III is presented in Section 5-6. Section 5 presents generic EFSM and RM models. Section 6 defines RM models for capability configuration management functionality. Related works are presented in Section 7, while Section 8 gives Summary and Conclusions.

2. Adaptability Properties

General and core functional adaptability properties are defined. Four *general properties are defined.* The framework must provide a flexible and common way of modeling services independent of the type of the service system, the service models must be based on mechanisms appropriate for the type of functionality, the software implementation mechanisms must be flexible and powerful, and there must be easy mapping of the conceptual service system models to the physical computing and communication platform. The *core functional adaptability properties* are grouped in three classes:

A): Rearrangement flexibility B): Failure robustness, and C): Resource load awareness and control

Rearrangement flexibility means that the system structure and the functionality are not fixed. Nodes, users, services, service components, capabilities, can be added, moved, removed according to needs. Mobility of persons, sessions, nodes, terminals is further seamlessly handled. New nodes and capabilities are found automatically when introduced, and needed information about changes is propagated. There is a continuous adaptation to changed environments and operation strategies/policies.

Robustness and survivability means that the architecture is dependable and distributed, and that the system can reconfigure itself in the presence of failures. Resources and functionality are duplicated, hardware and software component failures must be detected and reconfiguration and reinitialization must take place during system operation.

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

Property A defines the basic flexibility features of the architecture. Property B and C set requirements to concepts and features that are needed as a foundation for the specification and implementation of functionality that can provide these properties.

3. Architecture Solution – Features and Concepts

3.1 Service and computing architecture dimensions

To meet the *general properties defined in Section 2,* the TINA architecture principle [2] is followed. The TAPAS service framework has a computing architecture dimension and a functionality architecture dimension. *The service architecture* shows the structure of services and service components. The *computing architecture* is a generic architecture for the specification and execution of any service. While the service architecture has focus on the service functionality independent of implementation, the computing architecture has focus on the modeling of functionality with respect to implementation, but independent of the nature of the service functionality. The properties of the computing architecture, however, are the fundament for the creation of services with needed adaptable networking services properties. *So the core functional properties are realized by a mixture of the features and concepts of the computing architecture and the functionality of the service architecture.*

3.2 QoS related concepts

3.2.1 Capability and capability performance

Capability was introduced in Section 1. Capability is needed to meet all the three core properties. Performance concepts, however, are needed to meet the failure robustness as well as resource load awareness and control core property. *Capabilities* can be classified into resources, functions and data. Examples are CPU, memory, transmission links, special hardware, and special programs and data.

Capability performance measures comprise: i) capability capacity measures, ii) capability state measures and iii) capability QoS measures, i.e. traffic and dependability measures. *Capability capacity measure* examples are transmission channel capacity, CPU processing speed and disc size. *Capability state measure* examples are number of connections, and the number that is waiting. Important *traffic performance measures* are transfer time, waiting time, throughput and utilization. Important *dependability performance measures* are availability and recovery time.

3.2.2 Service performance

Service performance measures are performance concepts related to the service provided to the service user. These measures comprise i) service system state measures and ii) service system QoS measures, i.e. traffic and dependability measures

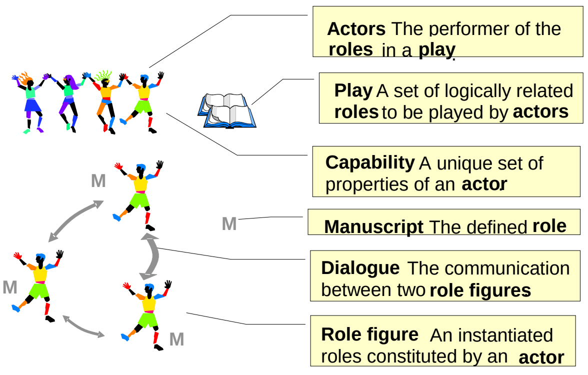
3.3.3 Service level agreements

Service level agreements (SLA) are agreements between the *service users* and the *service provider*. The agreement can contain elements such as: required capabilities, required service performance, payment for the service when the agreed QoS is offered and payment for the service in case of reduced QoS.

3.3 Computing Architecture

3.3.1 The theatre metaphor

The computing architecture is founded on a theatre metaphor (Figure 1). Actors perform roles according to manuscripts. Actors are software components in the nodes that can download manuscripts defining the roles to be played. An actor will constitute a *role figure* by behaving according to a manuscript. A *play* consists of several *actors* playing different *roles*, each possibly having different requirements on *capabilities*.



Actors The performer of the **roles** in a **play**
Play A set of logically related **roles** to be played by **actors**
Capability A unique set of properties of an **actor**
Manuscript The defined **role**
Dialogue The communication between two **role figures**
Role figure An instantiated roles constituted by an **actor**
M
M
M
M

Figure 1 - The theatre metaphor

3.3.2 Computing architecture viewpoints

The computing architecture is illustrated in Figure 2. For simplicity, the service user and service level agreements (SLA) are not included in the figure. The architecture has three views: the *service* view, *play* view and *physical* view. The service view considers the service as constituted by conceptual service components. The leaf conceptual service components are constituted by role figures which are implemented by actors. The actors are executed as operating system software components in nodes, which are physical processing units such as servers, routers, switches and user terminals.

The three view model is not a layered service model. A service system and its service components have models and model parameters related to the service view, play view and physical view. Capability and capability performance measures defined in the physical view, however, are visible in the play view as well as the service view. The service performance measures and SLA defined in the service view are also visible in the play view. This is not explicitly illustrated in Figure 2.

The *core platform* is a platform supporting the execution of service functionality based on the computing architecture functionality.

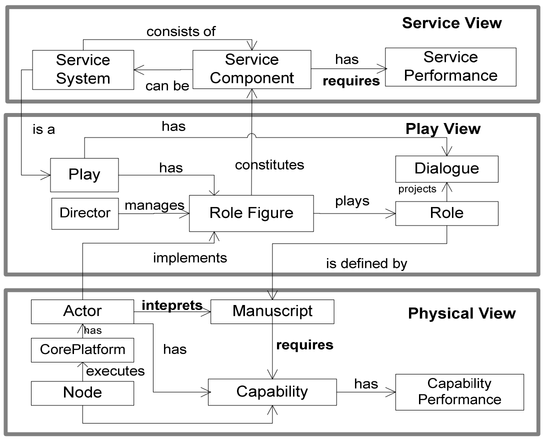


Figure 2 - TAPAS computing architecture

The computing architecture concepts and functionality are intended to be a basis for designing functionality that can meet the *general* as well as the *core properties*. The three views meet the general architecture requirements. Concerning the core properties, the play view concepts are primarily rearrangement flexibility oriented. The QoS concepts, gives a basis for functionality that can meet the *robustness and survivability* as well as the *QoS awareness and resource control* properties.

3.3.3 Actor and role figure types

An actor can be an EFSM or RM (See Section 1). Actors are able to download and execute EFSM- and RM-based manuscripts. Networked services are traditionally based on EFSM-based specification and execution. RM makes policy-based specification and operation possible. Policy-based software has a specification style, which is expressive and flexible.

A policy is a set of rules with related actions. A *policy system* is a set of policies. An RM is using a policy system to manage the behavior of a target system. A *static policy system* has a non-changeable set of policies, while a *dynamic policy* system has a changeable set of policies.

There are two types of role figures: i) EFSM-based role figures and ii) RM-based role figures. The EFSM-based role figure is constituted by an EFSM actor. The RM-based role figure is constituted by an RM actor.

4. Capability Configuration Management

Executing service components are instances of service component types. Service components constituted by EFSM-based and RM-based role figurers are denoted as *EFSM-based* and *RM-based* service components, respectively. EFSM-based service components will have requirements with respect to capability and service performance to perform their intended functionality (Figure 3).

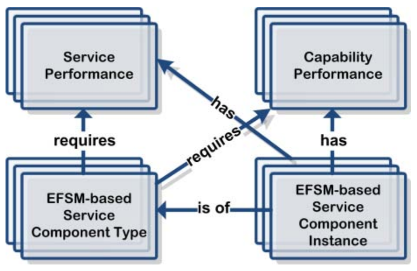

Service Performance
Capability Performance
has
requires
requires
has
EFSM-based Service Component Type
is of
EFSM-based Service Component Instance

Figure 3 – EFSM-based part of service system

These requirements are denoted as *required* capability and service performance. The capabilities and service performance of an executing service system are denoted as *inherent* capabilities and service performance. *System performance* is now defined as the sum of capability performance and service performance.

Capability configuration management is defined as the allocation, re-allocation and de-allocation of capabilities and comprises: i) service system capability initialization and re-initialization and ii) capability allocation adaptation. Service system capability initialization and re-initialization is the allocation of the capabilities for the service components to be distributed and instantiated. Capability allocation adaptation is the monitoring of the performance of the executing service system and the reallocation of capabilities within the executing service systems. In situations when the instantiated service systems are unable to adapt satisfactorily, capability configuration management can initiate a service system capability re-initialization for a redistribution and re-instantiation of the service system(s).

The service functionality architecture will need a rich set of functionality components to fulfill the core properties A)-C). On one hand, this paper has focus on the computing architecture features and concepts needed as basis for the general and core adaptability properties. On the other hand the paper has focus on the capability configuration management, which is an important adaptability function needed to utilize the adaptation potential of the capability-based computing architecture.

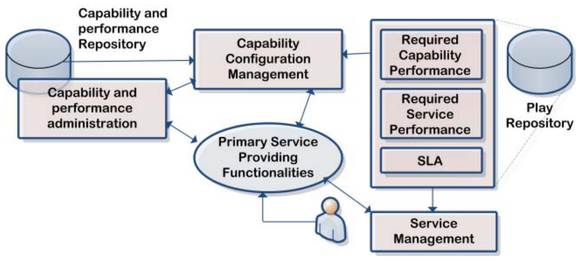

Capability and performance Repository
Capability and performance administration
Capability Configuration Management
Primary Service Providing Functionalities
Required Capability Performance
Required Service Performance
SLA
Play Repository
Service Management

Figure 4 – Some components of the service functionality architecture

The main components of the service functionality architecture related to capability configuration management are illustrated in Figure 4. In addition to the *primary service provisioning functionality*, there must be functionality for management of the networked service systems. This *management functionality* comprises: i) *service management. i.e.:* service specification, implementation, deployment, invocation, exhibition, discovery and access, ii) *capability and performance administration, i.e.:* monitoring and registration of available and allocated capabilities, monitoring and registration of system performance, and the provisioning of a view of available capabilities, capability performance and system performance, and iii) *capability configuration management*, with functionality as defined above.

5. Service Component Models

5.1 General

This section presents service component models. Section 0 presents generic EFSM and RM models. Section 0 presents the reasoning procedure applied by reasoning-based service components. The application domain for reasoning based service components and the integration of EFSMs and RMs in reasoning clusters are presented in Section 5.4.

5.2 EFSM and RM models

The following concepts are defined:

E	Functionality set of an EFSM type
$\hat{E}$	Functionality set of an EFSM instance
$\mathcal{R}$	Functionality set of a RM type
$\hat{\mathcal{R}}$	Functionality set of a RM instance
M	Manuscript type (EFSM or RM type)
$\hat{M}$	Manuscript instance (EFSM or RM instance)
C	Capability performance measures set
$\hat{C}_R$	Required capability performance set of an EFSM-based service component type
$\hat{C}_I$	Inherent capability performance set of an executing EFSM-based service component type
$\hat{C}_A$	Set of available capabilities in nodes
S	Service performance measures set
$\hat{S}_R$	Required service performance set for a EFSM-based service component type
$\hat{S}_I$	Inherent service performance set of an executing EFSM-based service component
I	Income functions set for the service components constituting a service. These functions will depend on the system performance.

An EFSM type E is defined (≡) as:

$$E \equiv \{ S_M, S_I, V, P, M(P), O(P), F_S, F_O, F_V \}, \quad (1)$$

where S_M is the set of states, S_I is the initial state, V is a set of variables, P is a set of parameters, $M(P)$ is a set of input signal with parameters, $O(P)$ is a set of output signal with parameters, F_S is the state transition function ($F_S = S \times M(P) \times V$), F_O is the output function, ($F_O = S \times M(P) \times V$) and F_V are the functions and tasks performed during a specific state transition such as computation on local data, communication initialization, database access, etc. A RM type $\mathcal{R}$ is defined as:

$$\mathcal{R} \equiv \{ \mathcal{Q}, \mathcal{F}, \mathcal{P}, \mathcal{T}, \mathcal{E}, \Sigma \} \quad (2a)$$
$$\mathcal{P} \equiv \{ \mathcal{X}, \mathcal{A} \}, \quad (2b)$$

where $\mathcal{Q}$ is the set of messages, $\mathcal{F}$ is a generic *reasoning procedure*, $\mathcal{P}$ is a policy system which consists of a set of *rules* $\mathcal{X}$ and a set of *actions* $\mathcal{A}$, $\mathcal{T}$ is a set of *system constraints* and $\mathcal{E}$ is a set of *facts*. The *reasoning procedure* is the procedure applied by RM to select the appropriate actions. Σ is a set of reasoning conditions defined by *trigger conditions* Σ_T, *and goal conditions* Σ_G. RM functionality is activated when a Σ_T is detected. When a trigger condition is true, the reasoning procedure transforms $\mathcal{Q}_1$ to $\mathcal{Q}_n$ by using $\mathcal{P}$ to match the system constraints $\mathcal{T}$ against the *facts* $\mathcal{E}$ and a set of suggest actions $\{\mathcal{A}_i, \mathcal{A}_j, \mathcal{A}_k \ldots \} \subseteq \mathcal{A}$.. The constraints rules and actions can have *variables*. The result of the reasoning can in addition to actions give instantiated variables.

The elements: $\mathcal{T}$, $\mathcal{E}$, Σ and $\mathcal{P}$ of an RM, depend on *the nature* of the reasoning service provided as well as the structural organization of EFSM, RM and capabilities. Section 5.4 will present reasoning models for cases that represents the functionality domain of the reasoning-based service components. These cases will be elaborated in Section 6.

5.3 Reasoning procedure

The reasoning procedure is based on *Equivalent Transformation* (*ET*) [6], which solves a given problem by transforming it through repetitive application of (semantically) equivalent transformation rules. Let $\mathbb{P}$ be a program which models an application for a knowledge base and $\mathcal{Q}_1$ is an initial message containing problems.

The semantics of $\mathbb{P} \cup \mathcal{Q}_1$ is denoted as $\mathcal{M}(\mathbb{P} \cup \mathcal{Q}_1)$. ET paradigm applies ET rules (procedural rewriting rules) in order to successively transform $\mathbb{P} \cup \mathcal{Q}_1$ into $\mathbb{P} \cup \mathcal{Q}_2$, $\mathbb{P} \cup \mathcal{Q}_3$, etc., while maintaining the condition $\mathcal{M}(\mathbb{P} \cup \mathcal{Q}_1) = \mathcal{M}(\mathbb{P} \cup \mathcal{Q}_2) = \mathcal{M}(\mathbb{P} \cup \mathcal{Q}_3) = \ldots$. Precisely, $\mathbb{P} \cup \mathcal{Q}_1$ is successively transformed until it becomes $\mathbb{P} \cup \mathcal{Q}_n$, where the message $\mathcal{Q}_n$ contains the list of *suggested actions* for the problem described by $\mathcal{Q}_1$. ET consists of sets of *ET rules* and ET *clauses* defined as follows:

ET clause: Head $\longleftarrow$ Body

ET rule: Head, Conditions $\longrightarrow$ Body

Head consists of one *head atom*, Body consists of body atoms and *Condition*s consists of condition atoms. A problem must be formulated as an ET clause. A *rule of a policy* as defined in Section 4.3 is modeled by an *ET rule*. The head atom is initially $\mathcal{Q}_1$. The *Head* will eventually contain $\mathcal{Q}_n$ if all the *body atoms* in the *Body* can be derived by *rules*. A body atom of a clause matching the head atom of a rule can be transformed into the rule's body atoms.

The reasoning procedure begins with a clause formulated by a message as follows:

$$\mathcal{Q}_1 \longleftarrow \mathcal{Q}_1 \qquad (3)$$

The meaning of (3) is that the Head $\mathcal{Q}_1$ is true when the Body $\mathcal{Q}_1$ is true. The goal of the reasoning procedure is to transform (3) until no body atom is left. Consider the following rule (4):

$$\text{Head , Conditions} \xrightarrow{ET} B_1, B_2, \ldots B_n. \qquad (4)$$

The rule (4) can transform the body atom $\mathcal{Q}_1$ of (3) into B_1, B_2, … B_n, provided that the body atom $\mathcal{Q}_1$ in (3) can match the Head in (4) and Conditions are not violated. Then, clause (3) will be transformed by (4) to (5) as follow:

$$\mathcal{Q}_2 \longleftarrow B_1', B_2', \ldots B_n'. \qquad (5)$$

During the transformation, variables in $\mathcal{Q}_1 \ldots \mathcal{Q}_n$ and the *list of suggested actions*, which are a subset of the actions $\mathcal{A}$ as defined in (2b), will be instantiated. The transformation of a clause ends when either 1) no body atom of the clause is left or 2) no rule can transform the remaining body atoms of the clause.

5.4 Functionality domain of reasoning machines

The functionality domain of RM-based service components can be separated into two cases. In Case I, the RMs of a service system has no access to system performance data of the service system. In this case, the RM provides an ordinary procedural service only. In Case II, RMs of the service system has access to system performance data of the service system and takes part in the configuration management functionality and in close cooperation with one or more EFSMs. For this case RM can be used as:

A. a procedural reasoning service for an EFSM-based role figure involving the access to system performance data for service system capability initialization and re-initialization

B. a feed-back loop service involving one or more EFSM-based role figures with access to system performance data for capability allocation adaptation

C. a feed-back loop service involving one or more EFSM-based and RM-based role figures with access to system performance data for dynamic changes of the policies for capability allocation adaptation as defined in B) above.

A *reasoning cluster* is an independent unit with respect to reasoning. It is a collection of EFSM-based service components with an associated *reasoning system* constituted by one or more *RM-based service components*. A reasoning cluster has associated: $\mathcal{Q}$, $\mathcal{P}$, $\mathcal{T}$, $\mathcal{E}$ and Σ and consider them as common data among EFSM-based and RM-based role figures. An RM must be activated by an EFSM. Depending of the nature of the cluster and the nature of the reasoning, a dedicated EFSM denoted as E_Σ will be needed to inspect the reasoning conditions and activate or deactivate the RM-based role figures within a reasoning cluster. Three reasoning cluster types (A, B and C) corresponding to the cases A-C defined above are illustrated in Figure 5.

In the cluster type A, the RM is a procedure invoked by an EFSM-based role figure. The EFSM-based role figure can set the reasoning condition observed by E_Σ The cluster type B represents a close interaction between RM-based role figures ($\mathcal{R}$) and EFSM-based role figures (E). An E_Σ manages the activation and deactivation of EFSM-based role figures. In the cluster type C, an RM-based role figure manages another RM-based role figure's policies.

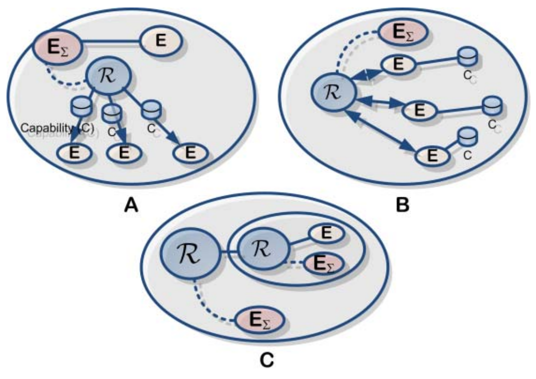


Figure 5 - Reasoning cluster types

6. Capability Configuration Management Functionality Models

In the following sections, reasoning models for the Cases A-C as defined in Section 6.4 will be elaborated from the generic RM type defined by (2a) and (2b) in Section 6.2.

6.1 Capability (re-)initialization

For this case, the indexes "init" and "re-init" indicates initialization and re-initialization. The constraints, facts and initialization condition can be expressed as follows:

$$\mathcal{T} \equiv \mathit{Expr}\{\, \mathrm{I}, (\hat{S}_{R,k}, \hat{C}_{R,k};\ k = [1, K])\} \quad (6)$$

$$\mathcal{E}_{\text{init}} \equiv \{\, \hat{C}_{A,n};\ n = [1, N_{\text{Node}}]\,\} \quad (7)$$

$$\mathcal{E}_{\text{re-init}} \equiv \{((\hat{S}_{I,lk}, \hat{C}_{I,lk};\ l = [1, L_k]),\ k = [1, K]), (\hat{C}_{A,n};\ n = [1, N_{\text{Node}}])\} \quad (8)$$

$$\Sigma_{\text{re-init}} \equiv \mathit{Expr}\{\, \hat{S}_{R,k}, \hat{C}_{R,k};\ k = [1, K]\,\} \quad (9)$$

$\mathcal{T}$ is some expression containing the income functions, and the required capability and service performance measures $\hat{C}_{R,k}$ and $\hat{S}_{R,k}$. The element $\mathcal{E}_{\text{init}}$, which contains the facts for the capability initialization case, contains the set of unallocated capabilities $\hat{C}_{A,n}$ of N_{Node} that can potentially contribute their capabilities for the EFSM-based role figures. The element $\mathcal{E}_{\text{re-init}}$ contains the facts for the capability re-initialization case that consists of the set of inherent capabilities performance measures $\hat{C}_{I,lk}$, the set of inherent service performance measures $\hat{S}_{I,lk}$ as well as the set of available unallocated capability $\hat{C}_{A,n}$.

The reasoning condition for the capability initialization as well as the E_{Σ} is under the supervision of an EFSM-based role figure. The content of $\mathcal{P}$ depends on the needed behavior of the adaptable service system. The components constituting $\Sigma_{\text{re-init}}$ are expressions of the states and the variables of the required capability and service performance measures $\hat{S}_{R,k}$ and $\hat{C}_{R,k}$ as given in (8d). A total specification of initial configuration and reconfiguration for a network printer example is given in [7].

6.2 Capability allocation adaptation

The constraints, facts and initialization condition can for this case be expressed similar to the equations for re-initialization (6), (8) and (9). The content of $\mathcal{P}$ depends on the needed behavior of the adaptable service system.

The feedback loop constituting a capability allocation adaptation is illustrated in Figure 6. An RM-based role figure called *Service System Adaptation Manager* $\mathcal{R}_1$ uses $\mathcal{E}$ as feedbacks from an EFSM-based role figure E. The rules $\mathcal{X}$ here are unchangeable. Uses of the expressions (6), (8) and (9) on a concrete capability allocation adaptation example using static policies are presented in [8].

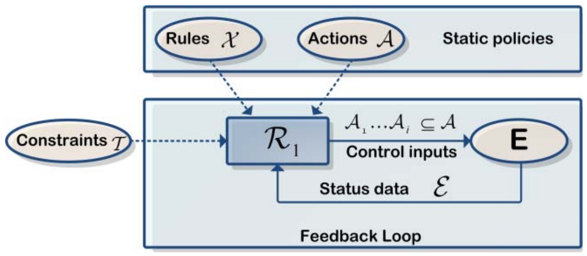

Rules $\mathcal{X}$
Actions $\mathcal{A}$
Static policies
Constraints $\mathcal{T}$
$\mathcal{R}_1$
$\mathcal{A}_1 \ldots \mathcal{A}_i \subseteq \mathcal{A}$
Control inputs
E
Status data $\mathcal{E}$
Feedback Loop

Figure 6 - Policy-based reasoning based on static policies

6.3 The adaptation of policies

The capability allocation adaptation using dynamic policies is illustrated in Figure 7. In addition to $\mathcal{R}_1$, a Policy Evaluator ($\mathcal{R}_2$) is added.

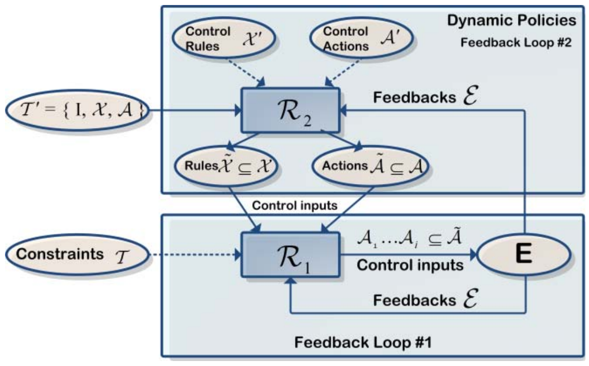

Control Rules $\mathcal{X}'$
Control Actions $\mathcal{A}'$
Dynamic Policies
Feedback Loop #2
$\mathcal{T}' = \{ \mathrm{I}, \mathcal{X}, \mathcal{A} \}$
$\mathcal{R}_2$
Feedbacks $\mathcal{E}$
Rules $\tilde{\mathcal{X}} \subseteq \mathcal{X}$
Actions $\tilde{\mathcal{A}} \subseteq \mathcal{A}$
Control inputs
Constraints $\mathcal{T}$
$\mathcal{R}_1$
$\mathcal{A}_1 \ldots \mathcal{A}_i \subseteq \tilde{\mathcal{A}}$
Control inputs
E
Feedbacks $\mathcal{E}$
Feedback Loop #1

Figure 7 - Policy-based reasoning based on dynamic policies

A generic rule-based reasoning system with dynamic policies can be extended from (2a) and (2b) as follows:

$$\mathcal{R}_1 \equiv \{ \mathcal{Q}, \mathcal{F}, \tilde{\mathcal{P}}, \mathcal{T}, \mathcal{E}, \Sigma \} \tag{10}$$

$$\tilde{\mathcal{P}} \equiv \{ \tilde{\mathcal{X}}, \tilde{\mathcal{A}} \} \tag{11}$$

$$\mathcal{R}_2 \equiv \{ \mathcal{Q}', \mathcal{F}, \mathcal{P}', \mathcal{T}', \mathcal{E}', \Sigma' \} \tag{12}$$

$$\mathcal{P}' \equiv \{ \mathcal{X}', \mathcal{A}' \} \tag{13}$$

$$\mathcal{T}' \equiv \{ \mathrm{I}, \mathcal{X}, \mathcal{A} \} \tag{14}$$

$\mathcal{Q}'$ is a set of internal messages between $\hat{\mathcal{R}}_1$ and $\hat{\mathcal{R}}_2$. The element $\mathcal{T}'$ consists of the set of income functions I and the set of total rules and actions ($\mathcal{X}$ and $\mathcal{A}$) of $\mathcal{R}_1$. The element $\mathcal{E}'$ is the same as $\mathcal{E}$ in 0. The element Σ' depends on the conditions for triggering the dynamic policies.

$\mathcal{P}'$ is a set of control policies consisting of the set of control rules $\mathcal{X}'$ and the set of control actions $\mathcal{A}'$. $\mathcal{X}'$ can use $\mathcal{A}'$ to re-order the priority, activate, de-activate and change the constraints of $\mathcal{X}$. As a result, the set of rules and actions used by $\mathcal{R}_1$ is now $\tilde{\mathcal{X}}$ and $\tilde{\mathcal{A}}$, which are derived from of $\mathcal{X}$ and $\mathcal{A}$.

The policy evaluator evaluates $\mathcal{X}$ at runtime based on *evaluation criteria, reference inputs* and *feedbacks*. Income functions are used as reference inputs, while the feedbacks are the system performance. Evaluation criteria can in general be history-based and prediction-based. The history-based evaluation determines the consequences of rules in the past based on system performance. The prediction-based evaluation determines the consequences of rules in the future based on mathematical equations represented by $\mathcal{X}'$. Dynamic policies need a certain period to evaluate the consequences of the rules used. Uses of the equations (10)-(14) on a concrete capability adaptation example using dynamic policies are presented in [8].

7. Related Work

Concerning service frameworks for adaptable service systems, focus is mostly on isolated features rather than on totality. Below we discuss related works that consider totality and that have rich feature sets.

The framework [9] has general architectural features and concepts and feedback loop for adaptability properties. However, it lacks a set of comprehensive features for capability configuration management. The work [10] has focus the capability allocation adaptation, but without considering general architectural features and concepts.

The architectures in [11] and [12] have a focus on agent-based computing architectures with feedback loops for autonomic service execution. However, it assumes that agents will have built-in service management functionality and all rely on some distributed algorithm. Likewise, [13] also share the same view while also describing a inclusive set of distributed protocols for adaptability properties.

Accord [14] is a framework with a comprehensive set of features including policies, and a computing architecture for service executions and configuration management. Physical capabilities, however, are not considered.

8. Summary and Conclusion

A capability-based service framework for adaptable service systems has been presented. The issues focused are I) Adaptability properties, II) Architecture solution and III) Capability configuration management functionality.

The *adaptability properties* defined are the basis for *architecture solution.* The *architecture* has a computing and a service architecture. The computing architecture has three views: the service view, the play view and the physical view. Further computing architecture features and concepts presented are:

- the theatre metaphor concepts (actor, role figure, capability, manuscript, role, play and dialogue)
- the ability of the Actor to download and interpret both EFSM-based and RM-based manuscripts
- QoS concepts (capability, service level agreement, capability and service performance)

Capability configuration management comprises service system capability initialization and re-initialization, and capability allocation adaptation. Generic models for EFSM-based and RM-based service components are presented. The application domain for reasoning machines within the context of capability configuration management are: a) capability initialization and re-initialization, b) capability allocation adaptation, and c) policy adaptation. RM-based service models for the cases a)-c) have been presented.

This paper, however, does neither contain data representation models nor concrete usage examples. The data representation of the QoS concepts, EFSMs and RMs are based on XML [8]. The aspects of capability configuration management that have been presented in this paper have been applied on concrete executable cases. A service system capability initialization and re-initialization example is presented in [7]. A capability allocation adaptation example is presented in [8].

References

[1] Y. Inoue, et al., The TINA Book: A Co-operative Solution for a Competitive World: Prentice Hall, 1999.

[2] D. L. Tennenhouse, et al., "A Survey of Active Network Research," IEEE Communications Magazine, vol. 35, 1997.

[3] J. O. Kephart and D. M. Chess, "The Vision of Autunomic Computing," Computer, vol. 36, pp. 41-50, 2003.

[4] P. Horn, "Autonomic Computing: IBMs Perspective on the State of Information Technology", http://www.research.ibm.com/autonomic/.

[5] F. A. Aagesen, et al., "Towards a Plug and Play Architecture for Telecommunications," Proceedings of IFIP TC6 Smartnet, Bankok, 1999.

[6] K. Akama, et al., "Solving Problems by Equivalent Transformation of Declarative Programs," Journal of the Japanese Society of Artificial Intelligence, vol. 13, pp. 944-952, 1998.

[7] F. A. Aagesen, et al., "Configuration Management for an Adaptable Service System," Proceedings of IFIP MAN, Ho Chi Minh City, Viet Nam, 2005.

[8] P. Supadulchai and F. A. Aagesen, "Policy-based Adaptable Service Systems Architecture," Proceedings of AINA-07, Niagara Falls, Canada, 2007.

[9] D. Garlan, et al., "Rainbow: Architecture-Based Self-Adaptation with Reusable Instrastructure," Computer, vol. 37, pp. 46-54, 2004.

[10] N. Samaan and A. Karmouch, "An Automated Policy-Based Management Framework for Differentiated Communication Systems," IEEE Journal on Selected Areas in Communications, vol. 23, pp. 2236-2247, 2005.

[11] D. Gracanin, et al., "Towards a model-driven architecture for autonomic systems," Proceedings of 11th IEEE ECBS, 2004.

[12] H. Tianfield, "Multi-agent based autonomic architecture for network management," Proceedings of IEEE INDIN, 2003, pp. 462 - 469.

[13] E. Vassev and J. Paquet, "Towards an Autonomic Element Architecture for ASSL," Proceedings of SEAMS '07, 2007.

[14] H. Liu and M. Parashar, "Accord: A Programming Framework for Autonomic Applications," IEEE Transactions on Systems, Man, and Cybernetics, vol. 36, pp. 341-351, 2006.

Paper H

Autonomous Production of Parameters of an Autonomous Capability Allocation Adaptation Model

Paramai Supadulchai and Finn Arve Aagesen

Prepared for a submission

Autonomous Production of Parameters of An Autonomous Capability Allocation Adaptation Model

Paramai Supadulchai and Finn Arve Aagesen

Department of Telematics
Norwegian University of Science and Technology (NTNU)
N7491 Trondheim, Norway
paramai@item.ntnu.no, finnarve@item.ntnu.no

Abstract— Autonomous capability allocation adaptation within a service framework for adaptable service systems is considered. The basis for autonomous capability allocation adaptation is service level agreements as well as service and capability performance. Capability allocation adaptation is based on capability allocation adaptation models. These models rely on fine-tuned parameters. This paper presents a framework for autonomous *parameter production* for capability allocation adaptation models. A parameter production example for capability allocation adaptation for an online music video on-demand service based on the proposed framework is given.

1. Introduction

Networked service systems are considered. Services are realized by service components, which by their inter-working provide a service in the role of a service provider to a service user. Service components are executed as software components in physical processing nodes, which are physical processing units such as servers, routers, switches, PCs and mobile phones. A *service framework* is here defined as the overall structural and behavior framework for the specification and execution of services.

An *adaptable service system* is defined as a service system which is able to adapt dynamically to changes in time and position related to users, nodes, capabilities, system performance, changed service requirements and policies.

This definition of adaptability covers a wide spectrum of functionality, and does also include autonomic communication [1, 2]. This paper has focus on adaptability with respect to capabilities. Capabilities are inherent properties of nodes used as a basis for the execution of services. Capabilities can be classified into resources, functions and data. Resource examples are CPU, memory, transmission links and special hardware.

Capability performance measures comprise: i) capability capacity measures, ii) capability state measures and iii) capability QoS measures, i.e. traffic and dependability measures. *Capacity* measure examples are transmission channel capacity and CPU processing speed. *State* measure examples are the number of connections and the number that is waiting. *Traffic* measure examples are waiting time, throughput and utilization. *Dependability* measure examples are availability and recovery time.

Capability configuration management is the allocation, re-allocation and de-allocation of capabilities. This functionality is arranged as 1) capability initialization, 2) capability allocation adaptation and capability re-initialization. Service system capability initialization and re-initialization are the allocation of the capabilities for the service components to be *physically distributed* and *instantiated*. Capability allocation adaptation is the monitoring of the performance of the executing service system and the reallocation of capabilities within the executing service systems. In situations when the instantiated service systems are unable to adapt satisfactorily, capability allocation adaptation can initiate a service system capability re-initialization for a *re-distribution* and re-instantiation of the service system(s).

The basis for capability initialization is predefined capability performance requirements. The basis for *capability allocation adaptation* and capability re-initialization are service level agreements (SLA) as well as capability and service performance. For definition of SLA and service performance, please see Section 2.2.

Capability allocation adaptation is based on capability adaptation models. These models need parameters derived from the service level agreements. These *capability allocation adaptation model parameters* can be produced. This paper presents some issues related to capability allocation adaptation. The work presented is related to TAPAS (TAPAS = Telematics Architecture for Play-based Adaptable Systems). See [3] and the URL: http://tapas.item.ntnu.no. The issues presented are:

I. Service framework and service component models
II. Autonomous capability allocation adaptation models
III. Autonomous parameter production for autonomous capability allocation adaptation models

The main contribution of this paper, however, is *Issue III*. Issue I is presented in Section 2 -3. TAPAS service framework has a *computing* and a *service* functionality dimension. The motivation for this separation as well as core QoS-related concepts are presented. The computing architecture is based on a theatre metaphor. An actor playing a role is either an Extended Finite State Machines (EFSM) or a Reasoning Machines (RM). Generic EFSM and RM models are presented in Section 3

Capability allocation adaptation models are presented in Section 4. Issue III is presented in Section 5 and 6. Related works are presented in Section 7, while Section 8 gives Summary and Conclusions.

2. Service Framework

2. 1 General

The motivation for and objectives of the TAPAS service framework are presented in [4]. The service framework intends to meet general and core functional adaptability properties. The core functional adaptability properties are i) rearrangement flexibility, ii) failure robustness, and iii) resource load awareness and control

The TAPAS service framework has a computing architecture dimension and a functionality architecture dimension. The service architecture shows the structure of services, i.e. application and management services. The service architecture, however, has focus on functionality independent of implementation,

The computing architecture is a generic architecture for the specification and execution of any service. The computing architecture has focus on the modeling of functionality with respect to implementation, but independent of the nature of the service functionality. The properties of the computing architecture are the fundament for the creation of services with needed core functional adaptability properties as defined above.

2.2 QoS-related concepts

Capability and capability performance was defined in section 1. Additional QoS-related concepts are:

- Service Performance
- Service Level Agreements

Service performance measures are performance concepts related to the service provided to the service user. The service performance measures comprise i) service system state measures and ii) service system QoS measures, i.e. traffic and dependability measures.

Service level agreements (SLA) are agreements between the service users and the service provider. The agreement can contain elements such as: required capabilities, required capability performance, required service performance, payment for the service when the agreed QoS is offered and payment for the service in case of reduced QoS. SLAs can further be categorized in *classes*, which represent distinctive service functionalities and performance requirements for a group of users with the same degree of satisfaction and cost.

2.3 The computing architecture

2.3.1 The theatre metaphor

The computing architecture is founded on a theatre metaphor. Actors perform roles according to manuscripts. Actors are software components in the nodes that can download manuscripts defining the roles to be played. An actor will constitute a role figure by behaving according to a manuscript. A play consists of several actors playing different roles, each possibly having different requirements on capabilities.

2.3.2 Computing architecture viewpoints

The computing architecture is illustrated in Figure 1. For simplicity, the service user and service level agreements (SLA) are not included in the figure.

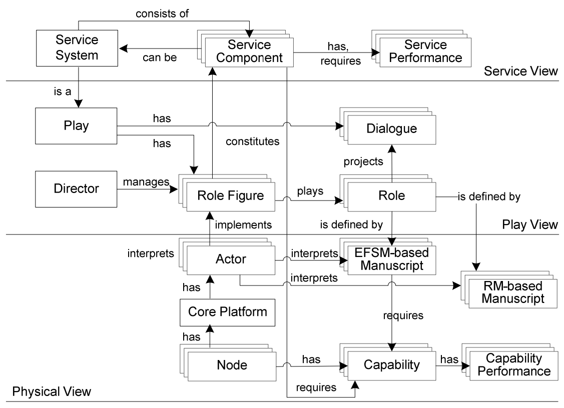


Figure 1 - TAPAS Computing Architecture

The architecture has three views: the service view, play view and physical view. The service view considers the service as constituted by conceptual service components. The leaf conceptual service components are constituted by role figures which are implemented by actors. The actors are executed as operating system software components in *nodes*, which are physical processing units such as servers, routers, switches and user terminals.

The three view model is not to be confused with a layered service model. A service system and its service components have models and model parameters related to the service view, play view and physical view. Capability and capability performance measures defined in the physical view, however, can be visible in the play view as well as the service view. The service performance measures and SLA defined in the service view are also visible in the play view. This is not explicitly illustrated in Figure 1.

The core platform is a platform supporting the execution of service functionality based on the computing architecture functionality.

2.3.3 Actor, role figure and service component types

An actor can be an extended finites state machine (EFSM) or reasoning machine (RM). Actors are able to download and execute EFSM- and RM-based manuscripts. Networked services are traditionally based on EFSM-based specification and execution. RM makes policy-based specification and operation possible. Policy-based software has a specification style, which is expressive and flexible.

In accordance with actor types there are also two types of role figures: i) EFSM-based role figures and ii) RM-based role figures. The EFSM-based role figure is constituted by an EFSM actor. The RM-based role figure is constituted by the combination of an EFSM actor and an RM actor.

Likewise, service components constituted by EFSM-based and RM-based role figurers are denoted as EFSM-based and RM-based service components, respectively. EFSM-based service components will have requirements with respect to capability and service performance to be able to perform their intended functionality (Figure 2). These requirements are denoted as *required* capability and service performance. The capability and service performance of an executing service system are denoted as *inherent* capability and service performance. *System performance* is defined as the sum of capability performance and service performance.

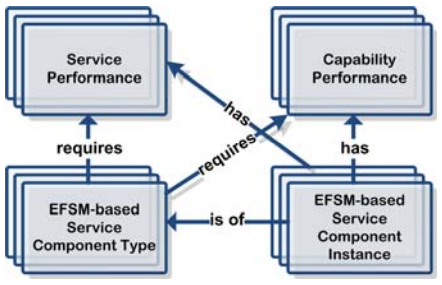

Service Performance
Capability Performance
has
requires
requires
has
EFSM-based Service Component Type
is of
EFSM-based Service Component Instance

Figure 2 - EFSM-based part of service system

3. Service Component Models

3.1 General

This section presents service component models. Section 3.1 presents generic EFSM and RM models. Section 3.2 presents the reasoning procedure applied by reasoning-based service components. The following concepts are defined:

Θ	Service level agreement
Ψ	Service offering priority
Φ	Agreed offered functionality of service component
Γ	Agreed performance
Λ	Payment in case of agreed performance,
Λ'	Payment in case of reduced performance
Λ''	Penalties when the SLA is broken; e.g. forced disconnection and exceeded waiting time
$I_\Delta(t)$	Income function during a time interval $(t, t+\Delta)$ as seen from the service provider; this function will depend on the system performance and SLAs
$J_\Delta(t)$	Cost functions set of physical nodes and capabilities during a time interval $(t, t+\Delta)$.

E	Functionality set of an EFSM-based type
Ê	Functionality set of an EFSM based instance
$\mathcal{R}$	Functionality set of an RM-based type
$\hat{\mathcal{R}}$	Functionality set of a RM-based instance
S	Service performance measure set
$\hat{S}_R$	Required service performance set
$\hat{S}_I$	Inherent service performance set
C	Capability performance measure set
$\hat{C}_R$	Required capability performance set
$\hat{C}_I$	Inherent capability performance set of all nodes
$\hat{C}_A$	Set of available unallocated capabilities in nodes

An SLA for a QoS class *q* "Θ*q*" is defined as:

$$\Theta q \equiv \{ \Psi q, \Phi q, \Gamma q, \Lambda q, \Lambda q' \} \qquad (3a)$$

The element I is the total income of the service system during a time interval between *t* and *t* + Δ, which can be calculated from the following equation:

$$I_\Delta(t) \equiv$$

$$\sum_{q}^{Q} (n_q'(t)\Lambda_q + n_q''(t)\Lambda_q' - n_q'''(t)\Lambda_q'') - J_\Delta(t) \qquad (3b)$$

where *Q* is the number of QoS classes, n_q is the number of service users in the QoS class *q*, n_q' is the number of service users in the QoS class *q* having the agreed performance, n_q'' is the number of service users in the QoS class *q* having the reduced performance, n_q''' is the number of service users in the QoS class *q* experiencing broken SLA ($n_q = n_q' + n_q'' + n_q'''$).

The formalism for the specification of the agreed functionality Φ is not considered in this paper. The agreed offered functionality is within the service system specified and implemented by EFSMs and RMs to be defined in following subsection.

3.2 EFSM and RM models

A generic EFSM-based actor type E is defined (≡) as:

$$E \equiv \{ S_M, S_I, V, P, M(P), O(P), F_S, F_O, F_V \}, \qquad (3c)$$

where S_M is the set of states, S_I is the initial state, V is a set of variables, P is a set of parameters, M(P) is a set of input signal with parameters, O(P) is a set of output signal with parameters, F_S is the state transition function ($F_S = S_M$ x M(P) x V), F_O is the output function, ($F_O = S_M$ x M(P) x V) and F_V are the functions and tasks performed during a specific state transition such as computation on local data, communication initialization, database access, etc.

A generic RM-based actor type $\mathcal{R}$ is defined (≡) as:

$$\mathcal{R} \equiv \{ \mathcal{Q}, \mathcal{F}, \mathcal{P}, \mathcal{T}, \Sigma, \mathcal{E}, \} \qquad (3d)$$

$$\mathcal{P} \equiv \{ \mathcal{X}, \mathcal{A} \}, \qquad (3e)$$

where $\mathcal{Q}$ is the set of transformation clauses, $\mathcal{F}$ is a generic *reasoning procedure*, $\mathcal{P}$ is a policy system which consists of a set of rules $\mathcal{X}$ and a set of actions $\mathcal{A}$. The quantities $\mathcal{T}$, Σ and $\mathcal{E}$ *are system constraints, reasoning conditions and facts* respectively. Σ consists of *trigger conditions* Σ_T *and goal conditions* Σ_G. The set {$\mathcal{T}$, Σ} of constraints and reasoning conditions are defined as the *parameters of the reasoning machine.*

The *reasoning procedure* is the procedure applied by RM to select the appropriate rules and actions. RM functionality is activated when a Σ_T is detected until a Σ_G is reached. When a trigger condition is true, the reasoning procedure transforms $\mathcal{Q}_i$ to $\mathcal{Q}_j$ by using $\mathcal{P}$ to match the system constraints $\mathcal{T}$ against the facts $\mathcal{E}$ and a set of suggest actions $\{\mathcal{A}_i, \mathcal{A}_j, \mathcal{A}_k\ldots\} \subseteq \mathcal{A}$. The constraints, rules and actions can have *variables*. The result of the reasoning can, in addition to actions, give instantiated variables. The reasoning procedure is explained more thoroughly in the following section 3.3.

An RM will need support for the continuous updating of $\mathcal{T}$, $\mathcal{E}$ and Σ, and for its activation and deactivation. $\mathcal{T}$, $\mathcal{E}$ and Σ are considered as common data for the supporting functionality and the RM.

3.3 Reasoning procedure

The reasoning procedure as defined above is based on *Equivalent Transformation* (*ET*) [5], which solves a given problem by transforming it through repetitive application of (semantically) equivalent transformation rules. Let $\mathbb{P}$ be a program which models an application for a knowledge base and $\mathcal{Q}_1$ is an initial transformation clause containing problems.

The semantics of $\mathbb{P} \cup \mathcal{Q}_1$ is denoted as $\mathcal{M}(\mathbb{P} \cup \mathcal{Q}_1)$. ET paradigm applies ET rules (procedural rewriting rules) in order to successively transform $\mathbb{P} \cup \mathcal{Q}_1$ into $\mathbb{P} \cup \mathcal{Q}_2$, $\mathbb{P} \cup \mathcal{Q}_3$, etc., while maintaining the condition $\mathcal{M}(\mathbb{P} \cup \mathcal{Q}_1) = \mathcal{M}(\mathbb{P} \cup \mathcal{Q}_2) = \mathcal{M}(\mathbb{P} \cup \mathcal{Q}_3) = \ldots$. Precisely, $\mathbb{P} \cup \mathcal{Q}_1$ is successively transformed until it becomes $\mathbb{P} \cup \mathcal{Q}_n$, where the final transformation clause $\mathcal{Q}_n$ contains the list of *suggested actions* for the problem described by $\mathcal{Q}_1$.

ET consists of sets of *ET rules* and ET *clauses* defined as follows:

$$\text{ET clause: Head} \longleftarrow \text{Body}$$

$$\text{ET rule: Head, Conditions} \longrightarrow \text{Body}$$

Head consists of one *head atom*, Body consists of body atoms and *Condition*s consists of condition atoms. A problem must be formulated as an ET clause. A *rule of a policy* as defined in Section 4.3 is modeled by an *ET rule*. The head atom is initially $H_{\mathcal{Q}1}$. The *Head* will eventually contain $H_{\mathcal{Q}n}$ if all the *body atoms* in the *Body* can be derived by *rules*. A body atom of a clause matching the head atom of a rule can be transformed into the rule's body atoms.

The reasoning procedure begins with a transformation clause formulated as follows:

$$\mathcal{Q}_1: \quad H_{\mathcal{Q}1} \longleftarrow H_{\mathcal{Q}1} \qquad (3f)$$

The meaning of (3f) is that the Head $H_{\mathcal{Q}1}$ is true when the Body $H_{\mathcal{Q}1}$ is true. The goal of the reasoning procedure is to transform (3f) until no body atom is left. Consider the following rule (3g):

$$\text{Head , Conditions} \xrightarrow{ET} B_1, B_2, \ldots B_n. \quad (3g)$$

The rule (3g) can transform the body atom $H_{\mathcal{Q}1}$ of (3f) into B_1, B_2, … B_n, provided that the body atom $H_{\mathcal{Q}1}$ in (3f) can match the Head in (3g) and Conditions are not violated. Then, clause (3f) will be transformed by (3g) to (3h) as follow:

$$\mathcal{Q}_2: \quad H_{\mathcal{Q}2} \longleftarrow B_1', B_2', \ldots B_n'. \qquad (3h)$$

During the transformation, variables in $H_{\mathcal{Q}1}\ldots H_{\mathcal{Q}n}$ and the *list of suggested actions*, which are a subset of the actions $\mathcal{A}$ as defined in (3e), will be instantiated. The transformation of a transformation clause ends when either 1) no body atom of the clause is left or 2) no rule can transform the remaining body atoms of the clause.

3.4 Functionality domain of reasoning machines

The functionality domain of RM-based service components can be separated into two cases. In Case I, the RMs of a service system has no access to system performance data of the service system. In this case, the RM provides an ordinary procedural service only. In Case II, RMs of the service system has access to system performance data of the service system and takes part in the configuration management functionality and in close cooperation with one or more EFSMs. For this case RM can be used as:

A. a procedural reasoning service for capability initialization and re-initialization based on capability performance measures
B. a feed-back loop service for capability allocation adaptation based on capability and system performance measures
C. a feed-back loop service for dynamic change of policies and the parameters as defined in Sec. 3.2 for the reasoning machines in case B

A *reasoning cluster* is an independent unit with respect to reasoning. It is a collection of EFSM-based service components with an associated *reasoning system* constituted by one or more *RM-based service components*. A reasoning cluster has associated: $\mathcal{Q}$, $\mathcal{P}$, $\mathcal{T}$, $\mathcal{E}$ and Σ and consider them as common data among EFSM-based and RM-based role figures. In addition to the EFSM-based service components in the cluster, a reasoning machine has an associated cooperating EFSM, which will invoke the reasoning machine. Depending of the nature of the cluster and the nature of the reasoning, a dedicated EFSM denoted as E_{Σ} can be needed to inspect the reasoning conditions and to activate the RMs associated EFSM. Three reasoning cluster types (A, B and C) corresponding to the functionality domain cases A-C defined above are illustrated in Figure 5.

In all cluster types, an RM-based *capability manager* $\mathcal{R}_{CM}$, gives suggested actions to an EFSM-based *capability manager* E_{CM} for managing capabilities.

In the cluster type A, $\mathcal{R}_{CM}$ is a procedural service for capability initialization and re-initialization. Re-initialization is initiated by E_{Σ}.

The cluster type B, $\mathcal{R}_{CM}$ is a part of a feed-back loop between the application service system and $E_{CM.}$ An E_{Σ} observes capability as well as service performance measures and manages the activation and deactivation of the capability management functionality, which is cooperation between E_{CM} and $\mathcal{R}_{CM}$.

In the cluster type C, another RM-based role figure $\mathcal{R}_{\mathcal{R}}$ manages the policies and parameters of $\mathcal{R}_{CM}$.

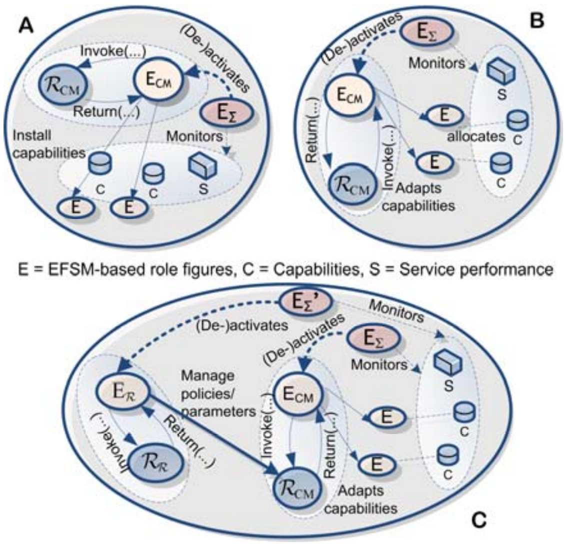


Figure 3 - Reasoning cluster types

4. Capability allocation adaptation

A feedback loop constituting capability allocation adaptation mechanism generally defined by (3d) and (3e) is illustrated in Figure . $\mathcal{R}_{CM}$ discussed in the previous section will now take the role of a *capability allocation adaptor*. When the service systems enter a triggering reasoning condition Σ_T, $\mathcal{R}_{CM}$ will be activated. The reasoning procedure is used to suggest a set of actions $\mathcal{A}$ that will lead the system back to a goal state Σ_G. $\mathcal{R}_{CM}$ is de-activated when service systems enter Σ_G.

The system constraints $\mathcal{T}$ and the reasoning conditions Σ_T, Σ_G are defined as the *parameters* of the capability allocation adaptation mechanism. The elements $\mathcal{P}$, $\mathcal{T}$, Σ_T, Σ_G, $\mathcal{E}$, $\mathcal{X}$ and $\mathcal{A}$ for the capability allocation adaptation model will now be abbreviated as $\mathcal{P}_{CM}$, $\mathcal{T}_{CM}$, $\Sigma_{CM,T}$, $\Sigma_{CM,G}$, $\mathcal{E}_{CM}$, $\mathcal{X}_{CM}$ and $\mathcal{A}_{CM}$.

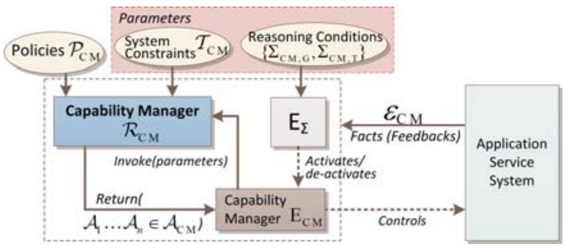

Parameters
Policies $\mathcal{P}_{CM}$
System Constraints $\mathcal{T}_{CM}$
Reasoning Conditions $\{\Sigma_{CM,G}, \Sigma_{CM,T}\}$
Capability Manager $\mathcal{R}_{CM}$
E_Σ
$\mathcal{E}_{CM}$
Facts (Feedbacks)
Application Service System
Invoke(parameters)
Activates/ de-activates
Return($\mathcal{A}_1 \ldots \mathcal{A}_n \in \mathcal{A}_{CM}$)
Capability Manager E_{CM}
Controls

Figure 4 - Capability Allocation Adaptation System

The system constrains, facts and reasoning conditions can generally be expressed as follows:

$$\mathcal{T}_{CM} \equiv Expr_{\mathcal{T}}\{\, I_\Delta(t), J_\Delta(t), (\hat{S}_{R,k}, \hat{C}_{R,k}, L_k;\ k = [1, K])\,\} \quad (4a)$$

$$\Sigma_{CM,T}, \Sigma_{CM,G} \equiv Expr_\Sigma\{\, \hat{S}_{R,k}, \hat{C}_{R,k};\ k = [1, K]\,\} \quad (4b)$$

$$\mathcal{E}_{CM} \equiv \{((\hat{S}_{I,lk}, \hat{C}_{I,lk};\ l = [1, L_k]), k = [1, K]),\ \ (\hat{C}_{A,n};\ n = [1, N_{Node}])\} \quad (4c)$$

$\mathcal{T}_{CM}$ is some expression containing the income and cost functions, and the required capability and service performance measures $\hat{C}_{R,k}$ and $\hat{S}_{R,k}$ for K service component types. The element $\mathcal{E}_{CM}$ contains the facts for the capability allocation adaptation that consists of the set of inherent capabilities performance measures $\hat{C}_{I,lk}$, the set of inherent service performance measures $\hat{S}_{I,lk}$ for all L_k instances of a service component type k as well as the set of available unallocated capability $\hat{C}_{A,n}$. The components constituting $\Sigma_{CM,G}$ and $\Sigma_{CM,T}$ are expressions of the triggering or goal states consisting of the required capability and service performance measures $\hat{S}_{R,k}$ and $\hat{C}_{R,k}$ as given in (4b). The policies $\mathcal{P}_{CM}$, constituted by $\mathcal{X}_{CM}$ and $\mathcal{A}_{CM}$ (see (3e), depend on the needed behavior of the adaptable service system.

The policies $\mathcal{P}_{CM}$ and parameters $\mathcal{T}_{CM}$ and Σ_{CM} of the capability adaptation model can be produced. This paper has focus on the parameters $\mathcal{T}_{CM}$ and Σ_{CM}. For the evaluation and production of $\mathcal{P}_{CM}$, it is referred to [6].

The parameters $\mathcal{T}_{CM}$ and Σ_{CM} are produced by a *parameter producer*. The capability allocation adaptation mechanism illustrated in Figure 4 will be extended with an RM-based $\mathcal{R}_{\mathcal{R}}$ applied in the reasoning cluster type C in the previous section. From now on $\mathcal{R}_{\mathcal{R}}$ has a role of a *parameter producer*.

5. Capability Allocation Model Parameter Production

5.1 Parameter Production Framework

The framework producing the parameters, i.e. the constraints and reasoning conditions is illustrated in Figure 5. The concepts in Figure 1 within the same level are grouped as the *service system*, *play system* and *actor system* respectively.

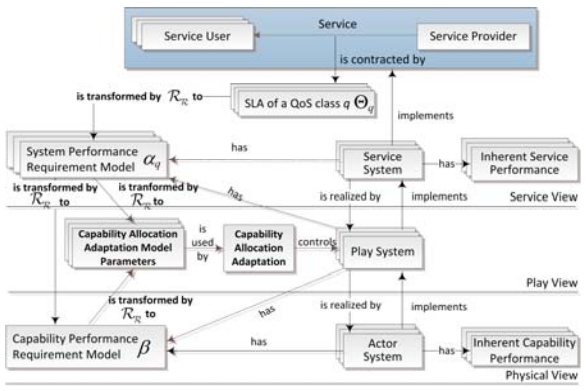

Service
Service User
Service Provider
is contracted by
is transformed by $\mathcal{R}_{\mathcal{R}}$ to
SLA of a QoS class q Θ_q
implements
System Performance Requirement Model α_q
has
Service System
has
Inherent Service Performance
is transformed by $\mathcal{R}_{\mathcal{R}}$ to
is tranformed by $\mathcal{R}_{\mathcal{R}}$ to
has
is realized by
implements
Service View
Capability Allocation Adaptation Model Parameters
is used by
Capability Allocation Adaptation
controls
Play System
Play View
is transformed by $\mathcal{R}_{\mathcal{R}}$ to
has
is realized by
implements
Capability Performance Requirement Model β
has
Actor System
has
Inherent Capability Performance
Physical View

Figure 5 – Capability allocation adaptation model parameter production framework

In Figure 5, the transformations of SLAs for the various QoS classes to the system and capability performance requirement models and to the capability allocation adaptation model parameters as illustrated by "*is transformed by* $\mathcal{R}_{\mathcal{R}}$ *to*" arrows are done by the parameter producer $\mathcal{R}_{\mathcal{R}}$. How the parameter producer works is illustrated in Figure 6. The system and capability performance requirement models α and β are the basis for the creation of the reasoning conditions Σ_{CM} and the system constraints $\mathcal{T}_{\mathrm{CM}}$ for capability allocation adaptation.

System performance requirement model α specifies the service view requirements of the various QoS classes consisting of service performance measures and the projection of capability performance measures in the service view. Capability performance requirement model β specifies the physical capability performance measure requirements resulting from the capability performance requirements of all QoS classes. Generic expressions for α and β can be given as:

$$\alpha_q \;=\; Expr_{\alpha}\,\{\, n_q,\, \hat{\mathrm{S}}_{\mathrm{R}q},\, \hat{\mathrm{C}}_{\mathrm{R}q} \,\} \qquad (5a)$$

$$\beta \;=\; \sum_{q}^{Q} Expr_{\beta}\,\{\, n_q,\, \hat{\mathrm{C}}_{\mathrm{R}q} \,\} \qquad (5b)$$

where $\hat{\mathrm{S}}_{\mathrm{R}q}$ and $\hat{\mathrm{C}}_{\mathrm{R}q}$ are the total set of required capability and service performance of n_q instances of a QoS class q. The elements $\$\alpha_x \in \alpha_q$ and $\$\beta_x \in \beta$ are denoted as system performance requirement model elements and capability performance requirement model elements respectively. The system performance requirement model elements from different SLA may belong to a certain type. The elements with the same type will affect the same capability performance requirements. This mechanism allows capabilities to be shared among different QoS classes.

Figure 6 illustrates the mechanism for the creation of α, β, $\mathcal{T}_{\mathrm{CM}}$ and Σ_{CM} from a given QoS class. The formalism in (3d) and (3e) are further supplemented with follow:

$$\mathcal{R}_{\mathcal{R}} \;\equiv\; \{\, \mathcal{Q}_{\mathcal{RR}}, \mathcal{F}, \mathcal{P}_{\mathcal{RR}}, \mathcal{T}_{\mathcal{RR}}, \mathcal{E}_{\mathcal{RR}}, \Sigma_{\mathcal{RR}} \,\} \quad (5c)$$

$$\mathcal{P}_{\mathcal{RR}} \equiv \{\, \mathcal{X}_{\mathcal{RR}}, \mathcal{A}_{\mathcal{RR}} \,\} \qquad (5d)$$

A parameter producer is an RM-based role figure that produces α, β, $\mathcal{T}_{\mathrm{CM}}$ and Σ_{CM}. The parameters $\mathcal{T}_{\mathrm{CM}}$ and Σ_{CM} will be used for the capability allocation adaptation. The element $\mathcal{Q}_{\mathcal{RR}}$ is the set of transformation clauses and variables that will 1) take an SLA as the input and produce α and β as the outputs and 2) take α and β as the inputs and produce $\mathcal{T}_{\mathrm{CM}}$ and Σ_{CM} as the outputs. $\mathcal{P}_{\mathcal{RR}}$ is the transformation policies consisting of the set of rules $\mathcal{X}_{\mathcal{RR}}$ and actions $\mathcal{A}_{\mathcal{RR}}$. The reasoning condition $\Sigma_{\mathcal{RR}}$ for the parameter producer itself is controlled by timer.

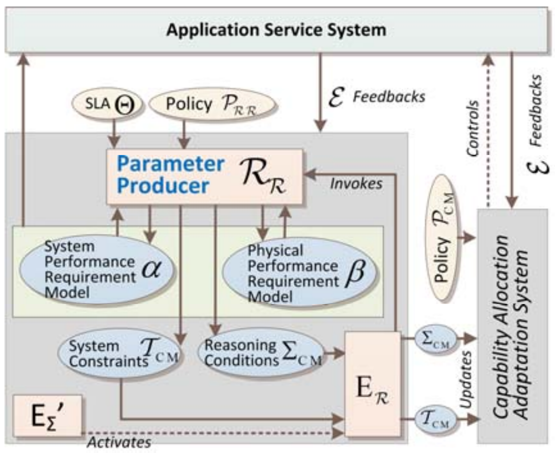

Application Service System
SLA Θ
Policy $\mathcal{P}_{\mathcal{RR}}$
$\mathcal{E}$ Feedbacks
Controls
$\mathcal{E}$ Feedbacks
Parameter Producer $\mathcal{R}_{\mathcal{R}}$
Invokes
Policy $\mathcal{P}_{\mathrm{CM}}$
System Performance Requirement Model α
Physical Performance Requirement Model β
System Constraints $\mathcal{T}_{\mathrm{CM}}$
Reasoning Conditions Σ_{CM}
Σ_{CM}
Updates
$\mathrm{E}_{\mathcal{R}}$
$\mathcal{T}_{\mathrm{CM}}$
Capability Allocation Adaptation System
E_{Σ}'
Activates

Figure 6 – Capability Allocation Adaptation Parameter Model Production

The facts $\mathcal{E}_{\mathcal{RR}}$ are the inherent capability and service performance measures, which are feedbacks from the service system being controlled by capability allocation adaptation. The formalism of $\mathcal{E}_{\mathcal{RR}}$ is the same as $\mathcal{E}_{\mathrm{CM}}$ given in (4c). During the initial configuration, the set $\mathcal{E}_{\mathcal{RR}}$ will contain only available capability performance measures of nodes ($\hat{C}_{A,n}$, n = [1, N_{Node}]).

The system constraints $\mathcal{T}_{\mathcal{RR}}$ containing the expression of SLAs, income functions, system and capability performance requirement models is defined as

$$\mathcal{T}_{\mathcal{RR}} \equiv \quad Expr\{\, \Theta, I_{\Delta}(t), J_{\Delta}(t), \alpha, \beta \,\} \qquad (5e)$$

The rules $\mathcal{X}_{\mathcal{RR}}$ for the production of α, β, $\mathcal{T}_{CM}$ and Σ_{CM} are presented in the following Section.

5.2 Rules for parameter production

$\mathcal{X}_{\mathcal{RR}}$ consists of six rules. Rules $\mathcal{X}_1$ and $\mathcal{X}_2$ are for the production of α. Rules $\mathcal{X}_3$ and $\mathcal{X}_4$ are for the production of β. Rules $\mathcal{X}_5$ and $\mathcal{X}_6$ are for the production of $\mathcal{T}_{CM}$ and Σ_{CM} respectively.

The structure of $\mathcal{X}_1$ and $\mathcal{X}_6$ follows the ET structure defined in 3.3. Variables in the rules prefixed with the \$ sign presents the inputs, outputs as well as intermediate transformation variables.

5.2.1 Production of α

The generic rules $\mathcal{X}_1$, $\mathcal{X}_2 \in \mathcal{X}_{\mathcal{RR}}$ for transforming an SLA $\$\Theta_q$ to a system performance requirement model $\$\alpha_q$ can be given as follow.

$$\begin{aligned}
\mathcal{X}_1(\,\$\Theta_q, \$\Phi_q \sim \$\alpha_q\,) \rightarrow \quad & B1.1{:}\ subset(\$E, \$\Phi_q\,), \\
& B1.2{:}\ \mathcal{X}_2\big(\$\Theta_q, \$E \sim \$\alpha_{qi}\big), \\
& B1.3{:}\ \$\alpha_q = \$\alpha_{qi} \cup \$\alpha_{qj}, \\
& B1.4{:}\ \mathcal{X}_1(\$\Theta_q, \$\Phi - \$E \sim \$\alpha_{qj}). \qquad (5f)
\end{aligned}$$

$$\begin{aligned}
\mathcal{X}_2\big(\$\Theta_q, \$E \sim \$\alpha_{qi}\,\big) \rightarrow \quad & \\
& B2.1{:}\ \$\hat{C}_{Rq} = dim_1(\$\Gamma_q, \$E, I_{\Delta}(t), \mathcal{E}_{\mathcal{RR}}, n_q), \\
& B2.2{:}\ \$\hat{S}_{R\,q} = dim_2(\$\Gamma_q, \$E, I_{\Delta}(t), \mathcal{E}_{\mathcal{RR}}, n_q), \\
& B2.3{:}\ \$\alpha_i = \{\, \$\alpha_x \mid \$\alpha_x \in \$\hat{S}_{Rq}, \$\hat{C}_{Rq} \,\}. \qquad (5g)
\end{aligned}$$

<u>Rule $\mathcal{X}_1$</u>

- $\mathcal{X}_1(\,\$\Theta_q, \$\Phi_q \sim \$\alpha_q\,)$ means that an SLA $\$\Theta_q$ with the functionality Φ_q can be transformed to $\$\alpha_q$ based on the calculation in the body atom *B1.1*, *B1.2*, *B1.3* and *B1.4*.
- *B1.1:* $subset(\$E, \Phi_q)$ looks for a service component \$E, which has offering functionalities as a subset of SLA's agreed functionality ($\$\Phi_q$).
- *B1.2:* $\mathcal{X}_2(\$\Theta_q, \$E \sim \$\alpha_{qi})$ calls rule $\mathcal{X}_2$ to determine a *partial* system performance requirement model $\$\alpha_{qi}$ of an \$E, which is the result of $subset(\$E, \Phi_q)$.
- *B1.3:* $\mathcal{X}_1(\$\Theta_q, \$\Phi_q - \$E \sim \$\alpha_{qj})$ call $\mathcal{X}_1$ again to find another partial system performance requirement model $\$\alpha_{qj}$ for the rest of the agreed functionality ($\$\Phi_q - \$E$).
- *B1.4:* $\$\alpha_q = \$\alpha_{qi} \cup \$\alpha_{qj}$ gives the system performance requirement model $\$\alpha_q$ which is the union of the partial sets $\$\alpha_{qi}$ and $\$\alpha_{qj}$.

<u>Rule $\mathcal{X}_2$</u>

- $\mathcal{X}_2(\$\Theta_q, \$\Phi_q \sim \$\alpha_{qi}\,)$ means that an SLA $\$\Theta_q$ with the functionality Φ can be transformed to a partial system performance requirement set $\$\alpha_{qi}$ based on the calculation in the body atoms *B1.1*, *B1.2*, *B1.3* and *B1.4*.
- *B2.1:* $dim_1(\$\Gamma_q, \$E, I_{\Delta}(t), \mathcal{E}_{\mathcal{RR}}, n_q) \in \mathcal{A}_{\mathcal{RR}}$ is a dimensioning function that determines the capacity of the required capability performance measures of a service component \$E based the agreed performance $\$\Gamma_q$ of an SLA class $\$\Theta_q$, the income function $I_{\Delta}(t)$, the fact $\mathcal{E}_{\mathcal{RR}}$ and the expected number of users n_q.
- Likewise, *B2.2:* $dim_2(\$\Gamma_q, \$E, I_{\Delta}(t), \mathcal{E}_{\mathcal{RR}}, n_q) \in \mathcal{A}_{\mathcal{RR}}$ is a dimensioning function that determines the capacity of the required service performance measures of a service component \$E.
- *B2.3:* $\$\alpha_{qi} = \{\, \$\alpha_x \mid \$\alpha_x \in \$\hat{S}_{Rq}, \$\hat{C}_{Rq} \,\}$ combines service and capability performance measure elements $\$\alpha_x$ (from $\$\hat{S}_{Rq}$ and $\$\hat{C}_{Rq}$) as a partial system performance requirement model set $\$\alpha_{iq}$.

5.2.2 Production of β

The generic rules $\mathcal{X}_3$, $\mathcal{X}_4 \in \mathcal{X}_{\mathcal{RR}}$ for transforming system performance requirement models $\$\alpha_q$; $q = [1, Q]$, into a capability performance requirement model $\$\beta$ can be given as follow.

$$\begin{aligned}
\mathcal{X}_3(\$\alpha \sim \$\beta) \rightarrow \quad & B3.1{:}\ \$\alpha_{x1} \in \$\alpha_1, \$\alpha_{x2} \in \$\alpha_2, \\
& \$\alpha_{x3} \in \$\alpha_3 \ldots
\end{aligned}$$

where $type(\$\alpha_{x1}) = type(\$\alpha_{x2}) = type(\$\alpha_{x2}) \ldots$,
B3.2: $\$\alpha_x = \$\alpha_{x1} + \$\alpha_{x2} + \$\alpha_{x3}$,
B3.3: $\mathcal{X}_4(\$\alpha_x, \$\beta_i)$,
B3.4: $\$\beta = \$\beta_i \cup \$\beta_j$,
B3.5: $\mathcal{X}_3(\$\alpha - \{\$\alpha_x\}, \$\beta_j)$. (5h)

$\mathcal{X}_4(\$\alpha_x \sim \$\beta_i) \longrightarrow$ *B4.1:* $\$\beta_i = dim_3(\$\alpha_x, \mathrm{I}_\Delta(t), \mathrm{J}_\Delta(t), \mathcal{E}_{\mathcal{RR}})$. (5i)

Rule $\mathcal{X}_3$

- The meaning of $\mathcal{X}_3(\$\alpha \sim \$\beta)$ is that the system performance requirement model $\$\alpha$ consisting of $\$\alpha_q$; $q = [1, Q]$, can be transformed to $\$\beta$ based on the calculation in the body atoms *B3.1*, *B3.2*, *B3.3* and *B3.4*.
- Let $\$\alpha_{x1}$, $\$\alpha_{x2}$, $\$\alpha_{x3}$,… be a system performance requirement model element from different SLA. *B3.1:* $\$\alpha_{x1} \in \$\alpha_1$, $\$\alpha_{x2} \in \$\alpha_2$, $\$\alpha_{x3} \in \$\alpha_3$ ... gets $\$\alpha_{x1}$, $\$\alpha_{x2}$, $\$\alpha_{x3}$,… elements from the system performance requirement models belonging to the same type ($type(\$\alpha_{x1}) = type(\$\alpha_{x2}) = type(\$\alpha_{x2})\ldots$).
- *B3.2:* $\$\alpha_x = \$\alpha_{x1} + \$\alpha_{x2} + \$\alpha_{x3}$ combines the system performance requirement model elements with the same type into $\$\alpha_x$, which is an intermediate system performance requirement model element of type *x*.
- *B3.3:* $\mathcal{X}_4(\$\alpha_x, \$\beta_i)$ calls rule $\mathcal{X}_4$ to find a *partial* capability performance requirement set $\$\beta_i$.
- *B3.4:* $\mathcal{X}_3(\$\alpha - \{\$\alpha_x\}, \$\beta_j)$ calls $\mathcal{X}_3$ again to find another partial $\$\beta_j$ for the rest of the capability and service performance measures ($\$\alpha - \{\$\alpha_x\}$).
- *B3.5:* $\$\beta = \$\beta_i \cup \$\beta_j$ combines the partial $\$\beta_i$ and $\$\beta_j$ as the capability performance requirement model $\$\beta$.

Rule $\mathcal{X}_4$.

- $\mathcal{X}_4(\$\alpha_x \sim \$\beta_i)$ means that $\$\alpha_x$, which is an intermediate system performance requirement model element of type *x*, can be transformed to a partial capability performance requirement model $\$\beta_i$ based on the calculation in the body atom *B4.1*.
- *B4.1:* The atoms $dim_3(\$\alpha_x, \mathrm{I}_\Delta(t), \mathrm{J}_\Delta(t), \mathcal{E}_{\mathcal{RR}}) \in \mathcal{A}_{\mathcal{RR}}$ is a dimensioning function that determines the capacity of the required physical performance measures constituting $\$\beta_i$ based on the given service performance requirement element $\$\alpha_x$, the income function $\mathrm{I}_\Delta(t)$, the cost of physical capabilities $\mathrm{J}_\Delta(t)$ as well as the available physical capabilities facts in $\mathcal{E}_{\mathcal{RR}}$.

5.2.3 Production of $\mathcal{T}_{CM}$

The generic rules $\mathcal{X}_5 \in \mathcal{X}_{\mathcal{RR}}$ for producing $\mathcal{T}_{\mathrm{CM}}$ from α and β can be given as

$\mathcal{X}_5(\{\$\Theta_q;\ q = [1,Q]\} \sim \$\mathcal{T}) \longrightarrow$
B5.1: $\$\Theta_i \in \{\$\Theta_q;\ q = [1,Q]\}$
B5.2: $\$\mathcal{T}_i = \$\alpha_i$,
B5.3: $\$\mathcal{T} = \$\mathcal{T}_i \cup \$\mathcal{T}_j \cup \$\beta$,
B5.4: $\mathcal{X}_5(\$\Theta_2, \$\mathcal{T}_j)$. (5j)

where $\$\alpha_i$ is the system performance requirement models of an SLA $\$\Theta_i$ and $\$\Theta_2 = \{\$\Theta_q;\ q = [1,Q]\} - \{\$\Theta_i\}$.

Rule $\mathcal{X}_5$

- $\mathcal{X}_5(\{\$\Theta_q;\ q = [1,Q]\} \sim \$\mathcal{T})$ means that the list of SLA $\{\$\Theta_i, \$\Theta_j\ldots\}$ can be transformed to $\$\mathcal{T}$ based the calculation in the body atoms *B5.1*, *B5.2*, *B5.3* and *B5.4*.
- *B5.1:* $\$\Theta_i \in \{\$\Theta_q;\ q = [1,Q]\}$ gets an SLA $\$\Theta_i$ from the list of SLA classes.
- *B5.2:* $\$\mathcal{T}_i = \$\alpha_i$ creates a *partial* system constraint set $\$\mathcal{T}_i$ as the SLA's associated system performance requirement models $\$\alpha_i$.
- *B5.3:* $\$\mathcal{T} = \$\mathcal{T}_i \cup \$\mathcal{T}_j \cup \$\beta$ combines the partial system constraint sets $\$\mathcal{T}_i$ and $\$\mathcal{T}_j$ and the capability performance requirement measure $\$\beta$ as the system constraints set $\$\mathcal{T}$.
- *B5.4:* $\mathcal{X}_5(\{\$\Theta_j\ldots\}, \$\mathcal{T}_j)$ calls $\mathcal{X}_5$ again to get the parameter $\mathcal{T}$ of the rest of SLAs.

5.2.4 Production of Σ_{CM}

Two types of reasoning conditions are considered: initial reasoning conditions and adapted reasoning conditions. The initial reasoning conditions are an *ordered list* of both system performance requirement model elements $\alpha_x \in \alpha$ and capability performance requirement model elements $\beta_x \in \beta$ of all SLA. Similar to $\mathcal{T}_{CM}$ production, the rules for producing the initial triggering reasoning conditions from α and β is given as follow:

$\mathcal{X}_6$ («\$$\Theta_i$, \$$\Theta_j$… » ~ \$$\Sigma_{CM,T}$) $\rightarrow$
B6.1: \$$\Theta_i \in$ «\$$\Theta_1$, \$$\Theta_2$… »
B6.2: \$$\Sigma_{CM,Ti}$ = {!\$$\alpha_x$ | \$$\alpha_x \in$ \$$\alpha_i$* },
B6.3: \$$\Sigma_{CM,T}$ = \$$\Sigma_{CM,Ti} \oplus$ \$$\Sigma_{CM,Tj} \oplus$ {!\$$\beta_x$ | \$$\beta_x \in$ \$$\beta$* },
B6.4: $\mathcal{X}_5$ («\$$\Theta_2$… », \$$\Sigma_{CM,Tj}$). (5k)

The set {…} and the set operation $\cup$ in $\mathcal{X}_5$ are replaced with an ordered list « … » and a list concatenation operation $\oplus$. This means that the capability allocation adaptation takes conditions with more priority (specified by SLA) before the conditions with less priority.

The system and capability performance measures \$$\alpha_i$* and \$$\beta$* are the subset of \$$\alpha_i$ and \$$\beta$ and contain only elements marked for reasoning conditions. {!\$$\alpha_x$ | \$$\alpha_x \in$ \$$\alpha_i$* } and {!\$$\beta_x$ | \$$\beta_x \in$ \$$\beta$* } mean each element of \$$\alpha_i$* and \$$\beta$* will be negated.

This algorithm gives more priority to \$$\alpha_x$ than \$$\beta_x$. The reason is because α generates β and it is likely that when a \$$\beta_x$ is triggered, some \$$\alpha_x$ will be also triggered.

An initial triggering goal condition is a negated pair of an initial triggering reasoning condition; e.g. an initial triggering reasoning condition $X > 0$ will be transformed into an initial triggering goal condition $X \leq 0$.

The examples in the next section consider only the initial triggering and goal reasoning conditions. However, the initial triggering and goal reasoning conditions can also be adapted. The adapted reasoning conditions will be observed from the consequences happening during a *capability allocation adaptation period*; i.e. when a triggering reasoning condition is fired until its paired goal condition is found. During this period, some service performance or capability performance will be observed and will be used as goodness criteria every observation period. Additional rules can be used to evaluate the goodness criteria and

- change the order of triggering reasoning conditions
- change the paired goal conditions.

6. Application Example

6.1 Scenario overview

A simple service system handling the capability management for a music video on-demand service is presented. The capability allocation adaptation and the capability allocation adaptation model parameter production will be presented. The intention is to illustrate the use of the parameter production framework

The service system is constituted by one or more media servers (MS) streaming media files to media players (MP) (Figure 7). There are two SLA classes, premium and ordinary, and Θ_P and Θ_O are the service level agreements between the service provider and two groups of users respectively. The SLA quantities for these two classes are given in Table 1.

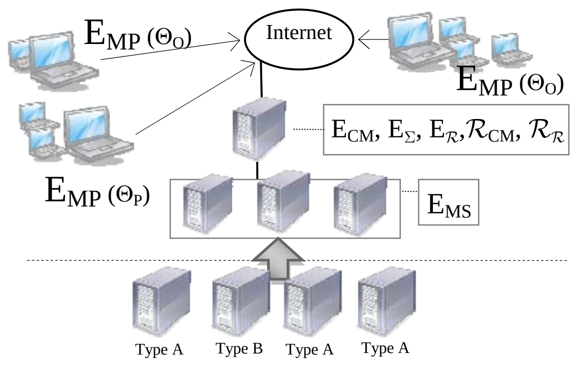


Figure 7 - A music video on-demand service system

The basic EFSM types constituting the music video application service system are a media server handler (E_{MS}) and a media player handler (E_{MP}). The EFSM types constituting a capability configuration management system are a capability manager (E_{CM}) and an E_{Σ}. According to the definition of capability configuration management in Section 0, capability initialization and re-initialization are not included in the example. This means that only the capability allocation adaptation is considered. E_{CM} is supported by two reasoning machines $\mathcal{R}_{CM}$ and $\mathcal{R}_{\mathcal{R}}$. $\mathcal{R}_{CM}$ has a role of the capability allocation adaptor and $\mathcal{R}_{\mathcal{R}}$ has a role of the parameter producer. $E_{\mathcal{R}}$ is the $\mathcal{R}_{\mathcal{R}}$'s associated EFSM.

Three different streaming throughput bit-rates (X) are offered: 500Kbps, 800Kbps and 1Mbps. The ordinary client connection throughput is 500 Kbps (X_O). The premium client connection throughput can be either 1Mbps (X_P) or 800 Kbps (X_P').

The unit applied for pricing is the price paid by an ordinary customer for one second streaming of the rate 500 Kbps. The payment when under normal operation for premium and ordinary users is Λ_P and $\Lambda_{O,}$ respectively. The payment when the system performance is reduced is Λ_P' and Λ_O', the penalty when waiting is Λ_{WP}'' and Λ_{WO}''. The penalty function for the disconnection of ordinary users is Λ_D''. The total income function during a monitoring time interval Δ of the system is $I_{\Delta}(t)$.

Table 1 – SLAs for the music video on-demand service

		Premium (Θ_P)	**Ordinary (Θ_O)**
Ψ	*Priority*	0	10
Φ	*Functionality*	Streaming	Streaming
Γ	*Streaming throughput*	1 Mbps or 800 Kbps	500 Kbps
	Maximum waiting time	0 sec	Best effort
Λ	*Payment when the system is under normal operation*	2 units	1 units
Λ'	*Payment when the performance is reduced*	1 units	0.5 units
Λ_W''	*Penalty for waiting*	10 units	5 units
Λ_D''	*Penalty for disconnection*	-	10 units

6.2 Capability allocation adaptation

The considered required capability for the E_{MS} role figure is Internet access link. Two type Nodes are applied for the execution of E_{MS}. Node type A has maximum access link capacity C_A = 100 Mbps, while Node type B has maximum access link capacity C_B = 1 Gbps.

The total access link capacity required ($C_{R,L}$) will be determined from the number of users and the set of agreed performance Γ. $C_{R,L}$ is the summation of the access link required by the premium users ($C_{R,LP}$) and the access link required by the ordinary users ($C_{R,LO}$).

When the required access link cannot be provided, a client may have to wait until some connected MPs have finished using the service. This will result in money payback to the waiting clients as illustrated in Table 1. An ordinary client can be disconnected, while a premium client may have to reduce the throughput. If a client is disconnected, the service provider pays a penalty.

The capability allocation rules are 1) to let the ordinary clients wait and let the premium clients use the service, 2) to reduce the premium clients' throughput and 3) to increase the number of servers and 4) to decrease the number of servers.

The mechanism to increase and decrease the number of servers, however, is controlled by two parameters $C_{R,LMin}$ and $C_{R,LMax}$. The capability manager cannot reduce the number of servers below the $C_{R,LMin}$ value and cannot increase the number of servers above the $C_{R,LMax}$ value.

The considered inherent capability performance measures $\hat{C}_I$ are the MS's inherent access link capacity ($C_{I,L}$).

The considered service performance measures $\hat{S}_I$ are the numbers of currently connected and waiting premium and ordinary clients ($n_{Con,O}$, $n_{Con,P}$, $n_{Wait,O}$, $n_{Wait,P}$), the number of currently disconnected ordinary clients ($n_{Dis,O}$), the number of MS (n_{MS}), the current total service time and waiting time of premium and ordinary clients ($T_{Serv,P}$, $T_{Serv,O}$, $T_{Wait,P}$, $T_{Wait,O}$). These values are observed per monitoring time interval Δ.

The system constraint parameters $\{\mathcal{T}_1 \dots \mathcal{T}_6\}$ for the capability allocation adaptation are the expression of the following service and capability performance measures:

$\mathcal{T}_1$: $T_{Wait,P} \leq t_{wp}$

$\mathcal{T}_2$: $T_{Wait,O} \leq t_{wo}$

$\mathcal{T}_3$: $C_{I,L} > C_{R,LMin}$

$\mathcal{T}_4$: $C_{I,L} < C_{R,LMax}$

$\mathcal{T}_5$: $n_A = n_{A\text{-}min}$

$\mathcal{T}_6$: $n_B = n_{B\text{-}min}$

The value of t_{wp}, t_{wo}, c_{min}, c_{max}, $n_{A\text{-}min}$ and $n_{B\text{-}min}$ will be calculated from the parameter production.

The set of actions $\mathcal{A}$ applied by the capability allocation adaptor is:

$$\mathcal{A} \equiv \{ \mathcal{A}_D, \mathcal{A}_B, \mathcal{A}_I, \mathcal{A}_R \} \quad (6a)$$

$\mathcal{A}_D$ *(Disconnect-Client)* tells MS to disconnect suggested MP_O. $\mathcal{A}_B$ *(Decrease-Bit-Rate)* tells MS to reduce throughput of suggested MP_P for a certain time period. $\mathcal{A}_I$ *(Initialize-Server)* tells MS to initiate a new MS of a certain node type, while $\mathcal{A}_R$ *(Remove-Server)* will remove an MS of a certain node type. The rule set $\mathcal{X}_{CM}$ for the capability manger is:

$$\mathcal{X}_{CM} \equiv \{ \mathcal{X}_D, \mathcal{X}_B, \mathcal{X}_I, \mathcal{X}_R \} \quad (6b)$$

$\mathcal{X}_D$ suggests $\mathcal{A}_D$ for disconnecting a list of suggested MP_O ***when*** $!\mathcal{T}_1$ AND $\Lambda_{WO} < \Lambda_{WP}$.

$\mathcal{X}_B$ suggests $\mathcal{A}_B$ for reducing throughput of a list of suggested MP_P ***when*** $!\mathcal{T}_2$ AND $\Lambda_{WO} > \Lambda_P - \Lambda_P'$.

$\mathcal{X}_I$ suggests $\mathcal{A}_I$ for initiating a new MS ***when*** $(!\mathcal{T}_1$ AND $!\mathcal{T}_2$ AND $\mathcal{T}_4$ AND $X_P \times N_{Wait,P} + X_O \times N_{Wait,O} > 0.1 \times C_{I,L})$ OR $(!\mathcal{T}_1$ AND $!\mathcal{T}_2$ AND $\mathcal{T}_4$ AND $(!\mathcal{T}_5$ OR $!\mathcal{T}_6))$.

$\mathcal{X}_R$ suggests $\mathcal{A}_R$ for removing an MS ***when*** $\mathcal{T}_1$ AND $\mathcal{T}_2$ AND $\mathcal{T}_3$ AND $X_P \times N_{Con,P} + X_O \times N_{Con,O} < 0.8 \times C_{I,L}$.

The symbols !, AND and OR are the logical operators *not*, *and* and *or* respectively.

Note that the number of MP_O to be disconnected by $\mathcal{X}_D$ is calculated by $N_{Wait,P} \times X_{P,1Mbps} / X_O$. In $\mathcal{X}_B$, the number of MP_P to decrease the bandwidth is calculated by $N_{Wait,O} \times X_O / (X_P - X_P')$.

6.3 Parameter production

6.3.1 System performance requirement model

The dimensioning functions dim_1 will be further specified as

```
function dim_1($\$\Theta.\Gamma, \$E, I_\Delta(t), \mathcal{E}_{\mathcal{RR}}, n_q$) {
    if ($\$\Theta == \Theta_P$) {    // Premium case
        $\$T_{Wait,P} = \Psi_P \times 1000$;
        return($T_{Wait,P} \leq \$T_{Wait,P}^*$);
    } else {    // Ordinary case
        $\$T_{Wait,O} = \Psi_O \times 1000$;
        return($T_{Wait,O} \leq \$T_{Wait,O}^*$);
    }
}                                        (6c)
```

The outputs of dim_1 are the total waiting time of the premium and ordinary clients ($T_{Wait,P}$ and $T_{Wait,O}$). According to Table 1, $T_{Wait,P}$ will always be less than zero while $T_{Wait,O}$ will be much larger value. These expressions are both marked for representing reasoning conditions.

The expected number of concurrent streaming premium and ordinary users are respectively n_{qP} and n_{qO} calculated from the mean number of users within the last 20 monitoring time interval (t-20Δ, t). The dimensioning functions dim_2 will be further specified as

```
function dim_2($\$\Gamma, \$E, I_\Delta(t), \mathcal{E}, n_q$) {
  if ($\$\Theta == \Theta_P$) {                 // Premium case
    if ($J_\Delta(t) \geq i_1$) {               // Cost is normal
      $\$C_{R,LP} = X_P \times n_{qP}$;         // Normal performance
    } else {                                    // Cost is too high
      $\$C_{R,LP} = X_P' \times n_{qP}$;        // Reduced performance
    }
    return ($C_{R,LO} > \$C_{R,LO}$);           // Return $\mathcal{T}$
  } else {                                      // Ordinary case
    $\$C_{R,LO} = X_O \times n_{qO}$;           // Normal performance
    return ($C_{R,LP} > \$C_{R,LP}$);           // Return $\mathcal{T}$
  }
}                                                                  (6d)
```

The outputs of dim_2 are the total capacity required for the premium and ordinary clients ($C_{R,LP}$ and $C_{R,LO}$). $I_\Delta(t) \geq i_1$ means the system has the total income equal or more than i_1, $I_\Delta(t) < i_1$ means the system has the total income less than i_1. Suppose $I_\Delta(t) \geq i_1$, the system performance requirement model α_P and α_O of the QoS classes Θ_P and Θ_O can be transformed by $\mathcal{X}_1$ and $\mathcal{X}_2$ as

$$\alpha_P = \{ T_{Wait,P} \leq 0^*,\ C_{R,LP} > X_P \times n_{qP} \} \quad (6e)$$

$$\alpha_O = \{ T_{Wait,O} \leq 10{,}000^*,\ C_{R,LO} > X_O \times n_{qO} \} \quad (6f)$$

6.3.2 *Capability performance requirement model*

The required *access link capacity* $C_{R,L}$ are considered. n_A and n_B are the number of instances of node type A and B for the execution of E_{MS} according to $C_{R,L}$.

The dimensioning function $dim_3(\$\alpha, I_\Delta(t), J_\Delta(t), \mathcal{E}_{\mathcal{RR}})$ will be further specified for the scenarios as

```
function dim_3($\$\alpha, I_\Delta(t), J_\Delta(t), \mathcal{E}_{\mathcal{RR}}$) {
if ($J < i_2$) {
    $\$n_B = \lceil L(C_{R,LP} + C_{R,LO}) / C_B \rceil$;
    $\$n_A = \lceil \{ L(C_{R,LP} + C_{R,LO}) - (n_B \times C_B) \} / C_A \rceil$;
} else {
    $\$n_A = \lceil L(C_{R,LP} + C_{R,LO}) / C_A \rceil$;
    $\$n_B = \lceil \{ L(C_{R,LP} + C_{R,LO}) - (n_A \times C_A) \} / C_B \rceil$;
}
$\$C_{R,LMax} = L(C_{R,LP} + C_{R,LO}) \times 1.2$;
$\$C_{R,LMin} = L(C_{R,LP} + C_{R,LO}) \times 0.8$;
return($n_A = \$n_A, n_B = \$n_B, C_{I,L} < \$C_{R,LMax},$
    $C_{I,L} > \$C_{R,LMin}$);
}                                                                  (6g)
```

where $L(\ )$ function determines the lower bound of the required link capacity L_R (e.g., $L(C_{R,LP} + C_{R,LO} > 5) = 5$), $\lceil\ \rceil$ is the always round up function (e.g., $\lceil 0.1 \rceil = 1$), $\{\ \}$ function gives zero when the argument is less than zero (e.g., $\{-5\} = 0$) and the unit is in Gbps.

The dim_3 function gives more priority to the node type B whenever the cost of node type B (J_B) is less than i_2. On the other hand, it gives more priority to Node Type A when $J_A \geq i_2$. The minimum and maximum values of $C_{R,L}$ ($C_{R,LMin}$ and $C_{R,LMax}$) are 20% less and more than the estimated capacity $L(C_{R,LP} + C_{R,LO})$.

Assume that $C_{R,LP} + C_{R,LO} > 1{,}000$ Mbps and the cost of server type B is less than < i_2, the capability performance requirement model β can be transformed by $\mathcal{X}_3$ and $\mathcal{X}_4$ as { $n_A = 1$, $n_B = 1$, $C_{I,L} < 800$ Mbps, $C_{I,L} > 1200$ Mbps}. Note that n_A and n_B are only the suggested numbers of node type A and B. It is allowed to have the number of n_A and n_B increased if needed. This will be determined by $C_{R,LMin}$ and $C_{R,LMax}$.

6.3.3 The system constraints $\mathcal{T}$

The performance requirements α_O, α_P and β will be transformed to the system constraints set $\mathcal{T}$ by (5j) as

{

$\mathcal{T}_1$: $T_{Wait,P} \leq 0$,

$\mathcal{T}_2$: $T_{Wait,O} \leq 10{,}000$,

$\mathcal{T}_3$: $L_I > 800$,

$\mathcal{T}_4$: $L_I < 1200$,

$\mathcal{T}_5$: $N_A = 1$,

$\mathcal{T}_6$: $N_B = 1$

}.

The reasoning conditions

The initial triggering reasoning conditions Σ_T will be transformed by (5k) as

«

Σ_{T1}: $T_{Wait,P} > 0$,

Σ_{T2}: $T_{Wait,O} > 10{,}000$,

».

And the paired initial reasoning goal condition Σ_G will be set as

«

Σ_{G1}: $T_{Wait,P} \leq 0$,

Σ_{G2}: $T_{Wait,O} \leq 10{,}000$,

».

As mentioned earlier, Σ_T and Σ_G will not be re-composed during the experimental simulation in the next section.

6.4 Results

The following results are aimed to compare the performance of the capability allocation adaptation with two parameter production cases. The first case is denoted as *dynamic parameters* where the parameters will always re-produced at run-time and the second case is denoted as *static parameters* where the parameters will be produced only once and will not be changed during run-time. The static parameter case has pre-defined initial parameters as given in 0 and 0.

The MP arrivals are modeled as a Poisson process with parameter λ_{QoS_Class}. The duration of streaming connections d_{QoS_Class} is constant. The quantity $\rho = ((\lambda_O \times d_O \times X_O) + (\lambda_P \times d_P \times X_P))/C_{I,AL}$ is the traffic offered to the MS access links. The MP_P arrival intensity is 15% of the total arrival intensity. The duration of streaming connections are set to 10 minutes, while the monitoring interval Δ is set to 1 minute. MPs stop waiting after 10 minutes. The income and penalty functions in units are given in Table 1.The cost for using an extra MS (J_A and J_B) can be either static ($J_{A\text{-}static}$ = 800 units/s per Node and and $J_{B\text{-}static}$ = 1600 units/s per Node) or variable as a function of time. The time with both values of J_A and J_B are at the same level is denoted as a J period.

Two additional scenarios are considered. The first scenario is the comparison of the total income when the system is operated by either static parameters or dynamic parameters when the value of ρ varies as a function of time while J_A and J_B are constant. In the second scenario, the value of ρ is constant while the values of J_A and J_B vary.

Figure 9 illustrates the accumulated total income of the first scenario. The time with ρ at a static level is denoted as the *ρ period*. The dashed line shows the variation of ρ, which can take the values 1, 0.5, 2 and 1.5 times of ρ = 1.44. The ρ period is $10 \times d_{QoS\_Class}$. The dynamic parameter case gives relatively better results. The static parameter case seems to need some *learning time* to fine tune the appropriate number of servers parameter at the beginning of each ρ period while in the dynamic case, this parameter is re-calculated.

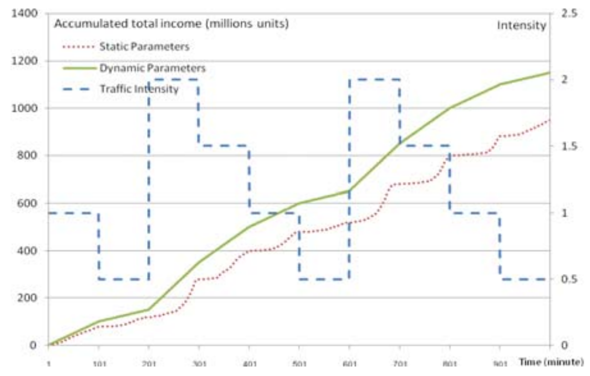


Figure 9 - the accumulated total income for dynamic traffic (ρ is varied)

Figure 10 illustrates the accumulated total income of the second scenario. The dotted line shows the variations of cost of the node type A and B (J_A and J_B), which are independent of each other. The dynamic parameter case has relatively less cost due to the ability to switch to cheaper combinations of node types when the cost functions are varied.

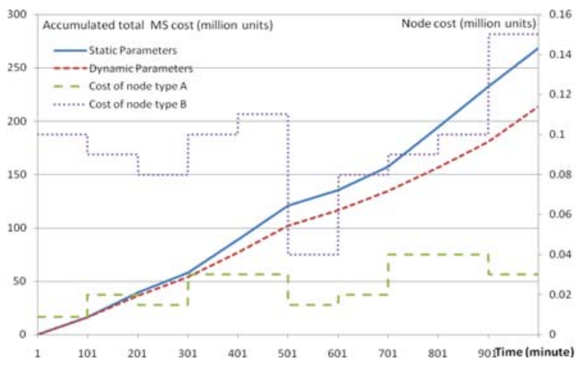


Figure 10 - the accumulated total income for variable cost (J is varied)

7. Related Work

The management of SLA and service performance in adaptable service systems has recently been addressed. Not only the adaptable service system can adapt to changes, the adaptable service system should also be able to satisfy the users as agreed on the SLA as well as maximize its performance. However, the work so far usualy deals with either algorithms in the physical layer [7, 8] or the management of SLA and service performance measures between OSI layers [9]. The transformation of SLA and service performance measures between conceptual levels has not really been the main focus.

However, the architecture in [10] shares the same vision of transforming high-level policy-based SLA into required capability performance in the physical level as well as the reasoning conditions to trigger the QoS manager for the resource adaptation. Unlike our architecture, the service configuration in [10], does not facilitate feedback loops that can dynamically changes allocation adaptation model parameters upon changing capability, service performance and income functions.

The work in [11] proposes a dynamic Internet pricing model based on a QoS architecture that is aimed to minimize network congestion as well as maximum the total income of the system. However, the methods used are slightly different from ours. The model in [11] charges users dynamically based on real usage as well as their willingness to pay while our architecture has penalty functions for the calculation of compensation when the SLA cannot be satisfied.

8. Conclusion

A capability allocation adaptation model parameter production framework for service configuration and QoS management of adaptable service systems has been presented. The framework handles QoS at three abstraction levels: the service, the play and the physical level. A service level agreement (SLA) class defines agreed service functionality, performance and payment for a group of service users with the same degree of satisfactions and cost. Service performances and capability performance requirement models describe the projection of the agreed service functionality and performance in the service and physical level. Policies have been introduced to increase flexibility and to provide a basis for optimality in the system specification and execution. Policies are used to control the service system according to the capability performance requirement models. As the main focus of this paper, the proposed architecture can transform SLA into the service and capability performance requirement models by using policies.

Two application cases handling the capability management of a music video on-demand service are presented. The intention is to illustrate the use of the proposed architecture. Case I uses static parameter while Case II uses dynamic parameters in the capability allocation adaptation. The use of dynamic parameters can be superior or equal to the use of static parameters. In the static parameters case, a set of capability adaptation model parameters may not likely be optimal for other system traffic load cases or cost functions. The production of capability adaptation model parameters can result optimal dimensioning.

The proposed architecture can also be a flexible tool for the experimentation with the parameters of capability allocation models with respect to optimization.

References

[1] P. Horn, "Autonomic Computing: IBMs Perspective on the State of Information Technology."

[2] FET, "Autonomic Communication Forums," 2007.

[3] F. A. Aagesen, P. Supadulchai, C. Anutariya, and M. M. Shiaa, "Configuration Management for an Adaptable Service System," in IFIP MAN, Ho Chi Minh City, Viet Nam, 2005.

[4] F. A. Aagesen, B. E. Helvik, U. Johansen, and H. Meling, "Plug and Play for Telecommunication Functionality -- Architecture and Demonstration Issues," in The International Conference on Information Technology for the New Millennium (IConIT), Thammasat University, Bangkok, Thailand, 2001.

[5] K. Akama, T. Shimitsu, and E. Miyamoto, "Solving Problems by Equivalent Transformation of Declarative Programs," Journal of the Japanese Society of Artificial Intelligence, vol. 13, pp. 944-952, 1998.

[6] P. Supadulchai and F. A. Aagesen, "Policy-based Adaptable Service Systems Architecture," in AINA-07, Niagara Falls, Canada, 2007.

[7] L. Chen and W. Heinzelman, "Network architecture to support QoS in mobile ad hoc networks," in The 2004 IEEE International Conference on Multimedia and Expo, 2004 (ICME '04), 2004, pp. 1715 - 1718.

[8] E. Bouillet and D. Mitra, "The Structure and Management of Service Level Agreements in Networks," IEEE Journal on Selected Areas in Communications, vol. 20, pp. 691-699, 2002.

[9] A. Iwata and N. Fujita, "A hierarchical multilayer QoS routing system with dynamic SLA management," IEEE Journal on Selected Areas in Communications, vol. 8, pp. 2603 - 2616, 2000.

[10] M. Ditze and T. Bresser, "Resource adaptation for audio-visual devices in the UPnP QoS architecture," in The 20th International Conference on Advanced Information Networking and Applications (AINA 2006), 2006.

[11] D. A. Vivanco, R. Q. Kemp, and A. P. Jayasumana, "Effectiveness of Internet Pricing Models over a QoS Architecture," in Proceedings. 28th Annual IEEE International Conference on Local Computer Networks (LCN '03), 2003, pp. 301-302.

Report I

NxET Reasoning Engine

Paramai Supadulchai

Plug-and-play Technical Report, Department of Telematics, NTNU

ISSN 1500-3868

NxET Reasoning Engine

Paramai Supadulchai

paramai@item.ntnu.no

Abstract

Native XML Equivalent Transformation (NxET) is an XML-based implementation of the ET paradigm (ET = Equivalent Transformation). NxET is designed for simplicity, modularity, extensibility, mobility and performance and can be used as a reasoning machine in various application domains. NxET has been used as supplemented reasoning functionality for the adaptable service systems based on TAPAS. This article focuses on NxET's computing paradigm, actor model, data model, and NxET implementation and usages. NxET's computing paradigm consists of a conceptual modeling language and a computation model. The actor model describes the use of NxET with actors in the TAPAS computing architecture. The data model describes XML-based data representations of the conceptual modeling language. The NxET implementation and usages illustrates NxET system components, processing mechanism and usages.

1. Background and motivation

Native XML Equivalent Transformation is an XML-based implementation of an ET paradigm (ET = Equivalent Transformation) written totally in Java. The work started in 2004 and the development of the main reasoning engine entered the maintenance state the mid of 2005. During this period, the author was the main developer. At the time the project was started, an implementation of XML-based ET (henceforward *the classic XET*) developed at Asian Institute of Technology [Anu02] had several short-comings.

- It was not designed for distributed or multi-agents systems. As a result, a configuration management system based on the classic XET must be centralized.
- Language features of the classic XET were limited and cannot be easily extended.
- The classic XET was an XML parser/converter that converts XET specification into ET specification that will be executed by an ET reasoning engine. The execution result is given back in an ET specification, which

will be converted back to XML. The conversion by the classic XET was not perfect and sometimes led to undesirable results.

NxET is designed based on the following principles.

- **Simplicity** – The reasoning functionality represented in XML-based representations can be directly executable by NxET. No further conversion is needed.
- **Modularity –** NxET can be used as supplemented reasoning functionality for an EFSM-based actor. Changing the behavior of an adaptable service system requires only modification of policies and parameters.
- **Extensibility** – NxET can be easily extended with new built-in functions and matching algorithms. NxET's built-in functions can be created by using Java Reflection.
- **Mobility –** NxET can be easily moved and deployed. The executable file are about 262 kilo-bytes in size and requires no special library besides the standard Java Runtime Environment (JRE).
- **Performance** –The NxET ability to process policies, constraints and facts using its native data structure increases performance of the reasoning, which also contributes to the overall service configuration performance.

NxET have been used in various application domains for solving problems that need an expressive data representation and a powerful computation mechanism. The application domains include, but are not limited to, adaptable service systems [Aag05], [Sup05a], [Sup05b], [Sup07a] and [Sup07b], database modeling [Bhr07], distributed application modeling [Nac07] and knowledge and ontology modeling [Rat07]. In present, NxET has been used and under further development at Asian Institute of Technology.

NxET has been related to Telematics Architecture for Adaptable Service Systems (TAPAS) and has been used as a supplemented reasoning functionality for the adaptable service systems based on TAPAS. This article focuses on NxET's computing paradigm, actor model, data model, and NxET implementation and usages.

NxET's computing paradigm is denoted as XET computing paradigm and consists of a conceptual modeling language and a computation model. The actor model describes the use of NxET with actors in the TAPAS computing architecture. The data model describes XML-based data representations of the conceptual modeling language. The NxET implementation and usages illustrates NxET system components, processing mechanism and usages. This article does not provide the TAPAS concepts, TAPAS service functionality architecture as well as TAPAS computing architecture. The readers should refer the first part of this thesis.

The rest of this article is structured as follows: Section 2 represents XET computational paradigm consisting of the conceptual language model and the computation model. Section 3 presents the actor model. Section 4 dicusses the data model supported by NxET. Section 5 describes NxET implementation and usages. Section 6 presents summary and conclusion.

2. XET Computational Paradigm

2.1 Conceptual language model

The conceptual modeling language is based on *XML Declarative Description (XDD)*, which extends ordinary well-form XML elements into XML expressions by incorporation of variables for an enhancement of expressive power and representation of implicit information. An XDD description is a set of XDD clauses, each of which is a formula of the form

XDD clause: $H \longleftarrow B_1, \ldots, B_m$ (1)

where m, $n \geq 0$, H and the B_i are head and body atoms. H atom is called the *head*, and $\{ B_1, \ldots, B_m \}$ the *body* of the clause. XDD clauses are just data representation. A computation model is needed. The XET computation model is based on *Equivalent Transformation* (*ET*) [Aka98], which solves a given problem by transforming it through repetitive application of (semantically) equivalent ET transformation rules. An XDD clause is modeled by an *ET transformation clause*.

The structures of an *ET transformation rule* and an ET *transformation clause* are defined as follows:

ET transformation clause: Head $\longleftarrow$ Body (2)

ET transformation rule: Head, Conditions $\longrightarrow$ Body (3)

Head consists of one *head atom*, Body consists of body atoms and *Conditions* consists of condition atoms. A problem must be formulated as an ET transformation clause. The head atom is initially $\mathcal{Q}_{h1}$. The *Head* will eventually contain $\mathcal{Q}_{hn}$ if all the *body atoms* in the *Body* can be derived by ET *transformation rules*. A body atom of a clause matching the head atom of a rule can be transformed into the transformation rule's body atoms.

2.2 Computation model

This section briefly describes the ET computation model. It is the procedure for selecting rules to be applied. Let $\mathbb{P}$ be a program which models an application for a knowledge base and $\mathcal{Q}_1$ is an initial ET transformation clause containing problems.

The semantics of $\mathbb{P} \cup \mathcal{Q}_1$ is denoted as $\mathcal{M}(\mathbb{P} \cup \mathcal{Q}_1)$. The ET paradigm applies ET transformation rules (procedural rewriting rules) in order to successively transform $\mathbb{P} \cup \mathcal{Q}_1$ into $\mathbb{P} \cup \mathcal{Q}_2$, $\mathbb{P} \cup \mathcal{Q}_3$, etc., while maintaining the condition $\mathcal{M}(\mathbb{P} \cup \mathcal{Q}_1) = \mathcal{M}(\mathbb{P} \cup \mathcal{Q}_2) = \mathcal{M}(\mathbb{P} \cup \mathcal{Q}_3) = \ldots$. Precisely, $\mathbb{P} \cup \mathcal{Q}_1$ is successively transformed until it becomes $\mathbb{P} \cup \mathcal{Q}_n$, where the message $\mathcal{Q}_n$ contains the list of *suggested actions* for the problem, which was described by $\mathcal{Q}_1$.

The reasoning procedure begins with an ET transformation clause formulated by as follows:

$$\mathcal{Q}_{h1} \qquad \longleftarrow \qquad \mathcal{Q}_{h1} \tag{4}$$

The meaning of (4) is that the Head $\mathcal{Q}_{h1}$ is true when the Body $\mathcal{Q}_{h1}$ is true. The goal of the reasoning procedure is to transform (4) until no body atom is left. Consider the following ET transformation rule:

$$\text{Head , Conditions} \qquad \longrightarrow \qquad B_1, B_2, \ldots B_n. \tag{5}$$

The rule (8) can transform the body atom $\mathcal{Q}_{h1}$ of (4) into $B_1, B_2, \ldots B_n$, provided that the body atom $\mathcal{Q}_{h1}$ in (4) can *match* the Head in (5) and Conditions are not violated. Then, clause (4) will be transformed by (5) to (6) as follow:

$$\mathcal{Q}_{h2} \qquad \longleftarrow \qquad B_1', B_2', \ldots B_n'. \tag{6}$$

During the transformation, variables in $\mathcal{Q}_1 \ldots \mathcal{Q}_n$ and the *list of suggested actions*, which are a subset of the actions $\mathcal{A}$ as defined in (6), will be instantiated. The transformation of an ET transformation clause ends when either 1) no body atom of the clause is left or 2) no ET transformation rule can transform the remaining body atoms of the clause.

The NxET's implementation of the computation model is denoted as the *reasoning procedure*.

3. The Actor Model

The actor role in the TAPAS computing architecture [Aag07] is defined as an Extended Finite State Machine (EFSM) extended with policies. The mechanism interpreting the manuscript is an EFSM interpreter extended with a reasoning functionality. A generic EFSM-based actor type E is defined (≡) as:

$$E \equiv \{ S_M, S_I, V, P, M(P), O(P), F_S, F_O, F_V \}, \tag{1}$$

where S_M is the set of states, S_I is the initial state, V is a set of variables, P is a set of parameters, M(P) is a set of input signal with parameters, O(P) is a set of output signal with parameters, F_S is the state transition function ($F_S = S_M$ x M(P) x V), F_O is the output function, ($F_O = S_M$ x M(P) x V) and F_V are the functions and tasks performed during a specific state transition such as computation on local data, communication initialization, database access, etc.

A generic RM-based actor type $\mathcal{R}$ is defined (≡) as:

$$\mathcal{R} \equiv \{ \mathcal{Q}, \mathcal{F}, \mathcal{P}, \mathcal{T}, \mathcal{E}, \Sigma \} \tag{2}$$

$$\mathcal{P} \equiv \{ \mathcal{X}, \mathcal{A} \}, \tag{3}$$

where $\mathcal{Q}$ is a set of transformation clauses, $\mathcal{F}$ is a generic *reasoning procedure* given in 2.2, $\mathcal{P}$ is a policy system which consists of a set of rules $\mathcal{X}$ and a set of actions $\mathcal{A}$. The quantities $\mathcal{T}$, Σ and $\mathcal{E}$ *are system constraints, reasoning conditions and facts* respectively. Σ consists of *trigger conditions* Σ_T *and goal conditions* Σ_G.

RM functionality is activated when a Σ_T is detected until a Σ_G is reached. When a trigger condition is true, the reasoning procedure transforms $\mathcal{Q}_i$ to $\mathcal{Q}_j$ by using $\mathcal{P}$ to match the system constraints $\mathcal{T}$ against the facts $\mathcal{E}$ and a set of suggest actions $\{\mathcal{A}_i, \mathcal{A}_j, \mathcal{A}_k \ldots \} \subseteq \mathcal{A}$. The constraints, rules and actions can have *variables*. The result of the reasoning can, in addition to actions, give instantiated variables.

A policy rule is modeled by an ET transformation rule in (3). A transformation clause is modeled by an ET transformation clause in (2).

4. Data model

The XML-based modeling of constraints, reasoning conditions and facts as well as the XML representation of the ET transformation rule and the ET transformation clauses are given in the this section, which is structured by the types of supporting data representation in the NxET data model. The data representation types of NxET can be XML variable, XML expression, XET clause and XET rule.

4.1 XML Variables

There are possible six disjoint XML variable types listed in Table 1. These variables can be specialized (or instantiated) into attributes names, element names, strings, zero or more attribute-value pair(s), one or more XML expression(s) and parts of XML expressions depending on their types. XML variables are begin with prefix (Nvar_, Svar_, Pvar_, Evar_, E1var_ and Ivar_) as given given in Table 1.

Table 1 – XML Variables

Type	Meaning and example specializations
N-variable (Nvar_)	*An element or attribute name*; e.g., <Nvar_element1>100</Nvar_element1> can be specialized into <Bandwidth unit="Mbps">100</Bandwidth>
S-variable (Svar_)	*A string*; e.g., <Bandwidth unit="Mbps">Svar_price</Bandwidth> can be specialized into <Bandwidth unit="Mbps">200</Bandwidth>
P-variable (Pvar_)	*A set of zero or more attribute-value pairs*; e.g., <Bandwidth Pvar_attrList="NULL">100</Bandwidth> can be specialized into <Bandwidth unit="Mbps">100</Bandwidth>
E-variable (Evar_)	*A sequence or a set of zero or more XML expressions*; e.g., <Bandwidth>Evar_priceDetails</Bandwidth> can be specialized into <Bandwidth><Min>20</Min><Max>200</Max>…</Bandwidth>

Type	Meaning and example specializations
E1-variable (E1var_)	*An XML expression*; e.g., <Network>E1var_priceDetails</Network> can be specialized into <Net-work><Bandwidth>…</Bandwidth></Network>
I-variable (Ivar_)	*Parts of XML expressions*; e.g., <Ivar_X><Bandwidth>100<Bandwidth></Ivar_X> can be specia-lized into <Node ID="1234">…<Bandwidth>100</Bandwidth>…</Node>

4.1 XML Expression

XML expressions are ordinary XML elements with variables in Table 1. However, an XML expression is not limited by one type of variable as given in the examples in Table 1. In fact, an XML expression can contain different types of variables; e.g. an XML expression <Element Pvar_attrList = "NULL"> E1var_X </Network> has both P- and E1-variables.

4.2 XET Clause

An XET clause is the XML-based representation of an ET transformation clause. The structure of an XET clause is illustrated as follows.

```
<xet:Clause>
  <xet:Head>…</xet:Head>
  <xet:Body>
    Body atom1, Body atom 2, …
  </xet:Body>
</xet:Rule>
```

An XET clause has a head and a body. The semantic of the head and body of the XET clause is the same as the semantic of the ET clause. The head consists of an XML expression. The body consists of several XML expressions, each of represents a body atom.

4.3 XET Rule

An XET rule is the XML-based representation of an ET transformation rule. The structure of an XET rule is illustrated as follows.

```
<xet:Rule xet:name="..." xet:priority="..." xet:class="...">
  <xet:Meta>...</xet:Meta>
  <xet:Head>...</xet:Head>
  <xet:Condition>...</xet:Condition>
  <xet:Body>
    Body atom1, Body atom 2, ...
  </xet:Body>
</xet:Rule>
```

An XET rule has a head, a condition and a body. The semantic of the head, condition and body of an XET rule is the same as the semantic of an ET rule. The head consists of an XML expression. The condition and body consists of XML expressions, each represents a condition atom or a body atom. The xet:Meta contains additional metadata for rules that will be used as additional processing instructions of the reasoning procedure.

xet:priority attribute specifies the priority of a rule. The priority represents the order of the rules to process. During the reasoning procedure NxET engine processes the rule having the most priority first. If the rule's head cannot be matched with the targeted body atom of the clause, the rule having the second most priority will be tried.

5. Implementation and usages

5.1 NxET system components

NxET system components are as follows.

- *NxET Parser* has a functionality to parse XML documents into XET Java objects.
- *NxET Executor* provides the main reasoning procedure and can invoke an XET Matcher and an XET Built-in manager for supplemented functionality.
- *NxET Matcher* provides a functionality for matching rules and transformation clauses. The matching tries all possibility to instantiate XML variables as listed in Table 1.
- *NxET Built-in Manager* provides the ability to invoke built-in functions.

- *NxET Configuration Manager* controls the behavior of *NxET Executor*, *NxET Parser*, *NxET Matcher* and *NxET Built-in Manager* based on the *NxET configuration*.
- *NxET Data Entities* consists of facts, constraints, policies and transformation clauses repository. Each repository stores relevant Java objects.
- *NxET Built-in Database* provides a database for storing NxET built-in functions.

5.2 NxET processing mechanism

The NxET system components and the processing mechanism are illustrated in Figure 1. Facts, policies, constraints, NxET configuration and an initial transformation clause in form of XML will be parsed and transformed into *NxET Java objects* by an *NxET Parser*. The Java objects will be kept separately in their own repository. An *NxET executor* fetches an initial transformation clause from the transformation clause repository and uses the reasoning procedure to select a transformation rule to apply. The matching of a transformation clause's body atom and rules' head atom is done by an *NxET Matcher*. The selected rule may refer to some constraints as well as some built-in functions. The NxET Executor will ask an *NxET Built-in Manager* to process the built-in functions and retrieve the constraints. An NxET Built-in Manager's built-in function is specified by a Java class, which extends the *Builtin* abstract class and instantiated by the Java reflection mechanism. [Kan06c] describes how to add an NxET Built-in function.

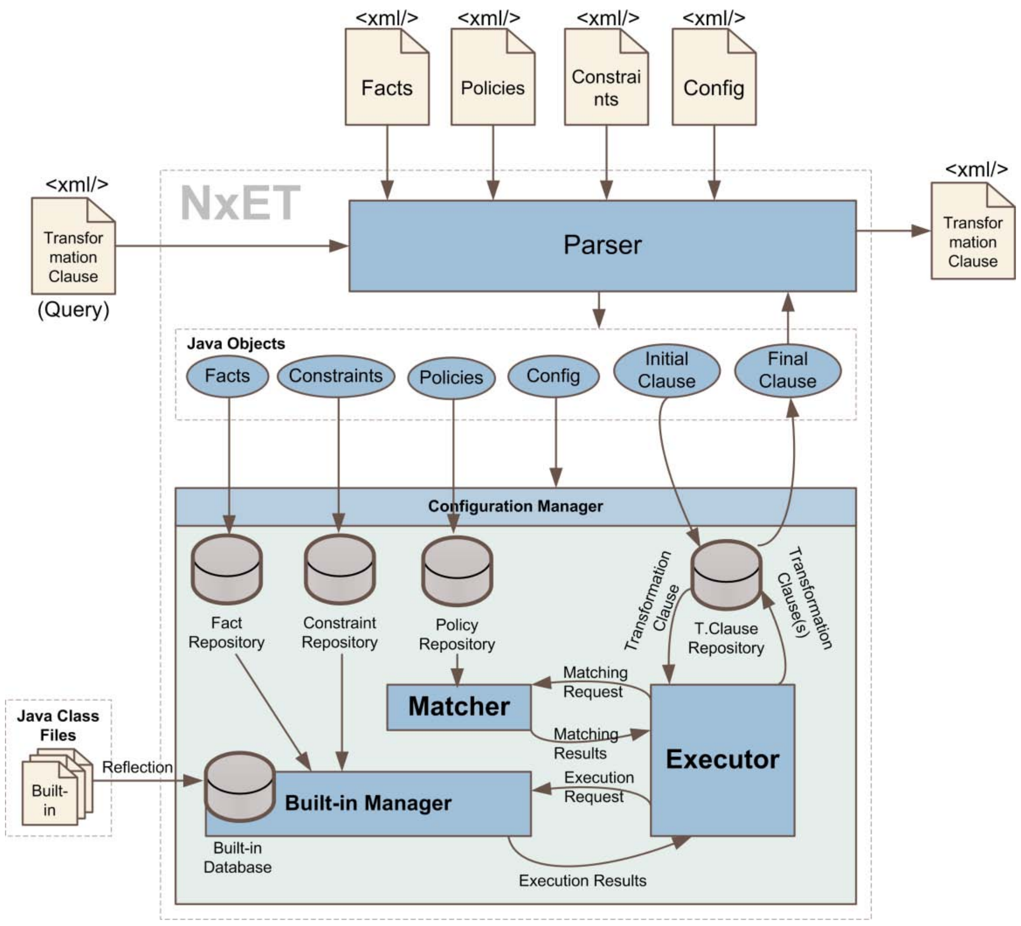


Figure 1 – NxET system components and processing mechanism

New clauses can be obtained as the result of a built-in function execution. The NxET executor will store the obtained clauses on the transformation clause repository. A new transformation clause will be processed by the NxET Executor when the present transformation clause processing has been finished.

When all clauses are processed, the NxET Executor sends the final clauses to the NxET Parser, which will transform the final Java object-based transformation clauses back into XML documents. The number of final transformation clauses indicates the number of answers derived by NxET.

NxET Configuration is used by an NxET configuration manager to control the behavior of the NxET Executor, NxET Matcher and NxET Built-in Manager.

5.3 NxET Usages

Generally, there are three ways to use NxET: 1) use NxET as a standalone Java application, 2) invoke NxET as a procedural call service within a Java application and 3) uses a GUI-based XET Rule Editor, which is a plug-in of the Protégé editor [Rat07]. The TAPAS actor model is based on method 2) where the TAPAS core platform uses NxET for a policy specification execution functionality. NxET can be invoked directly from an EFSM-based role figure. An NxET installation documentation is provided by [Kan06b] and an NxET developer's guide can be found in [Kan06a]. A list of NxET Built-in functions can be found in [Sup05c].

5.3.1 NxET as a stand-alone application

Figure 1 illustrates the use of NxET as a stand-alone application. Facts, constraints, policies, NxET configuration and an initial transformation clause must be expressed in XML documents. The NxET Parser is needed to process the XML documents.

5.3.2 NxET as a procedural call service within a Java application

When NxET is used within a Java application, e.g. as a supplemented reasoning functionality of the actor model of TAPAS, facts, constraints, policies, NxET configuration and the initial transformation clause can be supplied directly in Java objects. The NxET Parser is not needed.

5.3.3 Using XET Rule Editor

An XET Rule Editor is implemented by [Rat07]. When using the XET Rule Editor, facts, constraints, policies, NxET configuration and the initial transformation clause must be expressed in XML documents, which is similar to using NxET as a stand-alone application.

6. Summary and conclusion

This article presents NxET, which is an XML-based implementation of ET paradigm written in Java. NxET was aimed as a replacement of the classic XET that had several short-coming. The design principles were simplicity, modularity, extensibility, mobility and performance. NxET have been used in various application domains for solving problems that need an expressive data representation and a powerful computation mechanism. The application domains include, but are not limited to, adaptable service systems database modeling, distributed application modeling and knowledge and ontology modeling. In present, NxET has been used and under further development at Asian Institute of Technology.

NxET has been integrated as a supplemented reasoning functionality of the TAPAS core platform. The actor model describes the formalism of RM-based role figures, which have a supplemented reasoning functionality for EFSM-based role figures.

The NxET conceptual modeling language is based on XML Declarative Description, which extends ordinary well-form XML elements into XML expressions by incorporation of variables for an enhancement of expressive power and representation of implicit information. However, XDD clauses are just data representation. A computation model is needed. NxET's computation model is based on Equivalent Transformation (ET). Using ET, an XDD clause is modeled by an ET transformation clause. ET transformation rules are used to transform these clauses. The transformation mechanism is described by the NxET reasoning procedure.

NxET data model is based on XML. The data representation of NxET can be XML variables, XML expression, XET clause and XET rule. XET clause is the XML-based representation of ET transformation clauses. An XET rule is the XML-based representation of an ET transformation rule.

NxET system components, processing mechanism and usages are given. The system components describe software components constituting NxET. The processing mechanism describes the implementation details of the reasoning procedure. The usages shows that NxET can be used as either a standard application, a procedural call service within a Java application and by using XET Rule Editor.

References

[Aag05] F.A. Aagesen, P. Supadulchai, C. Anutariya and M.M. Shiaa, *Configuration Management for Adaptable Service Systems*, in Proceedings of IFIP International Conference on Metropolitan Area Network, Architecture, Protocols, Control and Management (MAN 2005), Ho Chi Minh City, Vietnam, 2005.

[Aag07] F.A. Aagesen and P. Supadulchai, *A Capability-based Service Framework for Adaptable Service Systems*, submitted to The 2nd International Conference on Advances in Information Technology (IAIT2007), Bangkok, Thailand, 2007.

[Aka98] K. Akama, T. Shimitsu, and E. Miyamoto, *Solving Problems by Equivalent*

Transformation of Declarative Programs, Journal of the Japanese Society of Artificial Intelligence, vol. 13, pp. 944-952, 1998.

[Anu02] Chutiporn Anutariya, Vilas Wuwongse, Vichit Wattanapailin, *An Equivalent-Tranformation-Based XML Rule Language*, RuleML 2002.

[Bhr07] T. Bhrammanee, V. Wuwongse, *Towards a Unified Representation Framework for Modelbases and Databases*, in Proceedings of the 9th International Conference on Decision Support Systems, Springer, 2007.

[Kan06a] Nattiya Kanhabua, *NxET Developer's Guide*, Technical Report, Asian Institute of Technology, 2006.

[Kan06b] Nattiya Kanhabua, *NxET Installation and User Manual*, Technical Report, Asian Institute of Technology, 2006.

[Kan06c] Nattiya Kanhabua, NxET – how to add a new built-in function, Technical Report, Asian Institute of Technology, 2006.

[Nac07] A. Naco, V. Wuwongse and C. Anutariya, *A transformation-based approach to application model development: class diagram generation*, Journal of International Journal of Software Engineering and Knowledge Engineering.

[Rat07] P. Ratanajaipan1, V. Wuwongse, E. Nantajeewarawat and C. Anutariya, XET Protégé Plug-in Environment, in Proceedings of the 10th International Protégé Conference, Budapest, Hungary, July 15-18, 2007.

[Sup05a] P. Supadulchai and F.A. Aagesen, *A Framework for Dynamic Service Composition*, in Proceedings of 1st International IEEE Workshop on Autonomic Communication and Computing (ACC 2005), Taormina, Italy, 2005.

[Sup05b] P. Supadulchai and F.A. Aagesen, *Autonomic Service Configuration by a Combined State Machine and Reasoning Engine-based Actor*, in Proceedings of the 2005 IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM 2005), Delta Centre-Ville Hotel, Montréal, Canada, 2005.

[Sup05c] Paramai Supadulchai and Nattiya Kanhabua, NxET built-in functions, Technical Report, Department of Telematics, NTNU, original 2005, modified version 2006.

[Sup07a] P. Supadulchai and F.A. Aagesen, *Policy-based Adaptable Service Systems Architecture*, in Proceedings of the IEEE 21st International Conference on

Advanced Information Networking and Applications (AINA-07), Niagara Falls, Canada, 2007.

[Sup07b] P. Supadulchai and F.A. Aagesen, *Towards Policy-Supported Adaptable Service Systems*, in Proceedings of the 13th Eunice Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services, University of Twente, the Netherlands, 2007.

Abbreviations

A1	Core functional adaptability property #1: rearrangement flexibility
A2	Core functional adaptability property #2: failure robustness
A3	Core functional adaptability property #3: Resource load awareness and control
C1	Thesis's contribution #1: Data model
C2	Thesis's contribution #2: Capability configuration management
C3	Thesis's contribution #3: Policy-based reasoning
C4	Thesis's contribution #4: Capability-based computing architecture
CCM	Capability Configuration Management
CIM	Common Information Model
CM	Capability Manager
CPU	Central Processing Unit
EFSM	Extended Finite State Machine
IN	Intelligent Network
ITEM	Department of Telematics, NTNU
NTNU	Norwegian University of Science and Technology
OWL	Web Ontology Language
PDA	Personal Digital Assistant
QoS	Quality of Service
RM	Reasoning Machine
RDF	Resource Definition Framework
RDFS	Resource Definition Framework Schema
SLA	Service Level Agreement
TAPAS	Telematics Architecture for Play-based Adaptable Service Systems
TINA	Tele-communication Information Networking Architecture

UniCS	Unified Capability and Status representation framework