Patcharee Thongtra

A Service Framework for Capability-based Adaptation in Adaptable Service Systems

Thesis for the degree of Philosophiae Doctor

Trondheim, September 2012

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

NTNU – Trondheim
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

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

ISBN 978-82-471-3798-7 (printed ver.)
ISBN 978-82-471-3799-4 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2012:246

Printed by NTNU-trykk

Abstract

Networked services are considered. A networked service is offered by a service system consisting of components, which by their interworkings provide a service in the role of a service provider to users. *An adaptable service system* is a service system that is able to adapt to changes brought about by users, nodes, capabilities, system performance, and service functionalities. A *capability* is defined as an inherent property of a node, which is used as a basis to implement a service. This thesis focuses on adaptability aspects related to nodes and capabilities.

A service framework is a system for the specification, management, and execution of adaptable service systems. Functionalities of the service framework enable various *adaptability types*, as follows: *capability-related adaptation*, *functionality-related adaptation*, and *context-related adaptation*. The capability-related adaptation is considered in this thesis. C*apability-related adaptation* is related to the shortage of capabilities with appropriate logical functionality or the overload or failure of capabilities leading to reduced system performance.

This thesis presents *a service framework for capability-related adaptation*. This service framework consists of *a computing architecture*, *concepts*, *ontologies*, *service component models*, *mechanisms*, *system architectures*, *and service execution platforms.* The service framework presented is related to TAPAS (Telematics Architecture for Play-based Adaptable Service Systems). The TAPAS concepts and architectures that existed at the beginning of this Ph.D. study are the foundation of this thesis. On the other hand, the research work presented in this thesis has contributed to new TAPAS concepts, ontologies, service component models, mechanisms, system architectures, and service execution platforms.

The service framework is classified into six contributions (**C1–C6**) as follows:

- **C1**: Ontology-based computing architecture
- **C2**: Ontologies, models, and representations
- **C3**: Policy-based reasoning
- **C4**: Autonomic-based model
- **C5**: Capability-related adaptation functionality realisation
- **C6**: Service execution platforms, prototypes, and simulations

Ontology-based computing architecture is an architectural framework for the specification and execution of adaptable service systems that provide any service functionality. *Ontologies, models, and representations* are applied for the formalisation and representation of the necessary concepts; these are also the basis for the formal definition of the requirements of the capabilities and service component specifications, i.e., the policy and goal specifications. *Policy-based reasoning* is a support functionality that takes adaptation actions based on flexible and expressive behavioural specifications. The *Autonomic-based model* is composed of the specification and learning mechanism of the Autonomic Elements that make up an autonomic service system, which is a specialised version of adaptable service systems. *Capability-related adaptation functionality realisation* defines specific functionalities, interworkings, and structural organisation of the service components that implement the functionalities required for capability-related adaptation. Finally, *service execution platforms, prototypes, and simulations* provide the validation and evaluation of the remaining contributions.

Preface

This thesis is submitted as partial fulfilment 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. The work presented in this thesis has been carried out under the supervision of Professor Finn Arve Aagesen.

This research work is related to the Telematics Architecture for Play-based Adaptable Service Systems (TAPAS). TAPAS aims at developing concepts, architectures, service execution platforms, and prototype implementations for adaptable service systems. TAPAS also aims to increase the efficiency of and to simplify the installation, deployment, operation, management, maintenance, and evolution of adaptable service systems. This project has been divided into several tasks, to which several participants have contributed.

This thesis is composed of two parts:

- ***Part I****: Overview*. Part I describes the background of this research, and gives an overview of Part II.

- ***Part II***: *Research papers.* Part II is a collection of papers. Each paper deals with one or more functionality required for the capability-related adaptation.

Part II, the main part of this thesis, consists of seven papers (Papers A–G), six of which have been published in the proceedings of international conferences. The remaining is a journal paper. These papers have been written in collaboration with other researchers, mainly my supervisor. In the papers where I am the first author, I have contributed to all parts of the paper, including the definition of the research hypothesis, defining formal concepts, models, or mechanisms, developing prototypes or simulation softwares, designing scenarios for validation and evaluation, conducting some experiments based on the scenarios, discussing the experimental results, evaluating the research hypothesis, writing the paper, and presenting the paper. In the papers where I am the second or the third author, my contributions are only in some parts of the paper.

Acknowledgements

I would like to first acknowledge Prof. Finn Arve Aagesen who supervised the research in this thesis. I thank him for his permanent accessibility, advice on request, and continuous support. I have learned from his rigorous work style and his strict requirements regarding the quality of research. My skills dealing with writing scientific papers have also improved much throughout my Ph.D. study. Thank you, Finn Arve, for your guidance.

I am grateful to NTNU for financial support, and to all the members of the TAPAS project. Special thanks go to Paramai Supadulchai who encouraged me to apply for the research fellowship, advised me on the TAPAS project, and helped me to settle down during my first few months in Norway. Many thanks go to Mazen Malek Shiaa for valuable knowledge and discussions on the TAPAS project, as well as to Shanshan Jiang for sharing her experiences during her study.

I am also grateful for all my colleagues at the Department of Telematics. Thank you very much, Linda Ariani Gunawan and Jing Xie, for sharing such a wonderful and memorable time with me. Natenapa Sriharee, thank you for your cooperation and feedback on research papers, as well as our discussions. Many thanks go to Pål Sturla Sæther and Asbjørn Karstensen for their technical support. My sincere thanks also go to Randi Schrøder Flønes and Mona Nordaune for their administrative support and advice, not only regarding my Ph.D. study, but also during my days at ITEM.

This thesis is dedicated to my family. I am very grateful for the continuous love and support from my parents, my brother Panomchai, and my sister Juthamas. I would also like to thank my close friends in Thailand, Patcharee Basu and Tippyarat Tansupasiri, who have always listened to my problems and supported me. In addition, thanks go to Ekapol Jeerangsuwan for his intelligent solutions to my programming problems. Last, but not least, my sincere thanks go to my husband Terje Sundet, for his continuous support, love, and motivation.

Patcharee Thongtra
April 2012
Trondheim, Norway

Contents

List of Figures

List of Tables

List of Papers

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

Table 1: Overview of the papers included in Part II.

Paper A	Ontology-based Capability Self-Configuration P. Thongtra, F.A. Aagesen, and N. Sriharee, in *Proc. 14th Eunice Open European Summer School on which networks for which services*, Brest, France, September 2008.
Paper B	Capability Ontology in Adaptable Service System Framework P. Thongtra and F.A. Aagesen, in *Proc. IARIA/IEEE 5th Int. Multi-Conf. Computing in the Global Information Technology (ICCGI 2010)*, Valencia, Spain, September 2010.
Paper C	OWL-based Node Capability Parameter Configuration P. Thongtra, F.A. Aagesen, and K. Dittawit, in *Proc. 18th EUNICE/IFIP WG 6.2, 6.6 Int. Conf. Information and Communications Technologies*, Budapest, Hungary, August 2012. *Lecture Notes in Computer Science (LNCS) 7479*, pp. 124-135, 2012.
Paper D	An Adaptable Capability Monitoring System P. Thongtra and F.A. Aagesen, in *Proc. IARIA/IEEE 6th Int. Conf. Networking and Services (ICNS 2010)*, Cancun, Mexico, March 2010.
Paper E	Towards Policy-Supported Adaptable Service Systems P. Supadulchai, F.A. Aagesen, and P. Thongtra, in *Proc. 13th Eunice Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services*, Twente, Netherlands, June 2007. *Lecture Notes in Computer Science (LNCS) 4606*, pp. 128-140, 2007.
Paper F	An Autonomic Framework for Service Configuration P. Thongtra and F.A. Aagesen, in *Proc. IARIA/IEEE 6th Int. Multi-Conf. Computing in the Global Information Technology (ICCGI*

	2011), Luxembourg, June 2011.
Paper G	On Capability-related Adaptation in Networked Service Systems F.A. Aagesen and P. Thongtra, *Int. Journal of Computer Networks & Communications (IJCNC)*, vol. 4, no. 4, July 2012.

Table 2: Overview of the paper published but not included in this thesis.

[1]	On Adaptability Issues of Networked Service Systems F.A. Aagesen and P. Thongtra, in *Proc. IEEE 2nd Int. Conf. Digital Information and Communication Technology and its Applications (DCTAP2012)*, Bangkok, Thailand, May 2012.

Abbreviations

ASN.1	Abstract Syntax Notation One
AA	The functionality component Adaptation administration
C1	Thesis contribution #1: Ontology-based computing architecture
C2	Thesis contribution #2: Ontologies, models, and representations
C3	Thesis contribution #3: Policy-based reasoning
C4	Thesis contribution #4: Autonomic-based model
C5	Thesis contribution #5: Capability-related adaptation functionality realisation
C6	Thesis contribution #6: Service execution platforms, prototypes, and simulations
CA	The functionality component Context Administration
CC	The functionality component Capability configuration
CCPS	Capability Configuration Process Specification
CIM	Common Information Model
CSA	The functionality component Capability and service administration
CSM	The functionality component Capability and service monitoring
CU	The functionality component Capability usage allocation
DMTF	Distributed Management Task Force
EFSM	Extended Finite State Machine
FD	The functionality component Fault diagnosis
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
II	The functionality component Installation and instantiation
IN	Intelligent Network
LM	Learning Mechanism
MIB	Management Information Base
NETCONF	Network Configuration Protocol
OWL	Web Ontology Language
PCIM	Policy Core Information Model
PD	The functionality component Performance diagnosis
PI	The functionality component Platform installation
PSC	Primary Service Component
QoS	Quality of Service
RM	Reasoning Machine
TAPAS	Telematics Architecture for Play-based Adaptable Service Systems
TINA	Telecommunication Information Networking Architecture
TMN	Telecommunication Management Network
SLA	Service Level Agreement
SM	The functionality component Service component movement
SMI	Structure of Managed Information
SNMP	Simple Network Management Protocol
SOAP	Simple Open Access Protocol
VLAN	Virtual Local Area Network
VTP	VLAN Trunking Protocol
WBEM	Web-Based Enterprise Management

XML	eXtensible Markup Language
XDD	XML Declarative Description language

Part I – Overview

1 Introduction

This section provides an introduction of the thesis. Section 1.1 presents general issues of adaptable service systems that are relevant to the research. Section 1.2 gives an outline of the thesis, and Section 1.3 presents a suggested paper reading guideline.

1.1 Adaptable service systems

General issues of adaptable service systems are described in the following subsections. Section 1.1.1 presents related definitions used in this thesis. Section 1.1.2 describes a service system life cycle. Section 1.1.3 describes various adaptability types, while Section 1.1.4 discusses closed feedback loop.

1.1.1 Related definitions

A networked service is a functionality offered by a *service system* consisting of service components, which by their interworking provide a service in the role of a service provider to service users. Service users can be human users as well as software and/or hardware components. *A service component* is executed as a software component in *nodes*, which are physical processing units such as servers, routers, switches, PCs, and mobile phones. A service system consists of service components; in turn, a service component can also be a service system, thus constituting a hierarchy of service systems.

Networked service systems have been an important research topic in the last few decades. During the 1980s and 1990s, the research was primarily focussed on flexibility and efficiency in the definition, design, deployment, and execution of the networked services. Example topics include IN (Intelligent Network) [ITU92-1], TMN (Telecommunication Management Network) [ITU92-2], TINA (Telecommunications Information Networking Architecture) [BDD99], Mobile Agents [Woo02], and Active and Programmable Networks [TSS97]. Since the mid 1990s, example topics also include Distributed Computing [CDK01], Peer-to-Peer Computing [MKL02], Web Services [Jos07], and Semantic Web [GFC03].

Due to the increasing complexity and heterogeneity in today's distributed computing systems, the research focus in this field has shifted towards *adaptability*, which deals with how the network service systems can adapt to *changes* in their environments. Example topics are Adaptable service systems as well as Autonomic systems and autonomic communications [DDF06].

In this thesis, *an adaptable service system* is defined as a service system that is able to adapt to changes in users, nodes, capabilities, system performance, and service functionalities.

A capability is an inherent property of a node that is used as a basis for implementing a service. Capabilities can be classified according to *capability types* and *capability parameters*. *A capability type* defines entities with common characteristics, while *a capability parameter* describes the characteristics of a capability type. Capability parameters are classified as *functionality parameters*, *performance parameters,* and *inference parameters*. *Functionality parameters* define functionality features, *performance parameters* define performance measures, and *inference parameters* define relations to other capability types. Capability performance parameters are further classified as *capacity parameters*, *state parameters,* and *QoS parameters*. *Capacity parameters* include transmission channel capacity, CPU processing speed, and disk size. Examples of *state parameters* are the number of available streaming connections and the number of packets discarded in a specific buffer. Examples of *QoS parameters* include capability throughput, capability utilisation, capability availability, and capability recovery time after errors.

The services provided to the service users can, in the same way as the capabilities, be described by *functional parameters* and *performance parameters*. Service performance parameters are further classified as *state parameters* and *QoS parameters*. *State parameters* include the number of users waiting to use the service and the number of users using the service. Examples of *QoS parameters* are service connection time, service transfer time, service waiting time, service throughput, service utilisation, service availability, and service recovery time after errors.

The functionality and performance parameters of the capability and service observed at one time instant or an interval are denoted as *inherent capability and service functionality* and *inherent capability and service performance*. *System performance* is defined as a common concept for capability performance parameters and service performance parameters.

An autonomic system is defined as a system that has the ability to manage itself and to adapt dynamically to changes in accordance with given objectives [KC03][WHW04]. An autonomic system consists of distributed components denoted as *autonomic elements*. According to this definition, an autonomic service system is a specialised adaptable service system, which imposes stricter requirements on the architecture solution in order to be approved as an autonomic solution. Autonomic systems will be discussed below in Section 6.

1.1.2 Service system life cycle

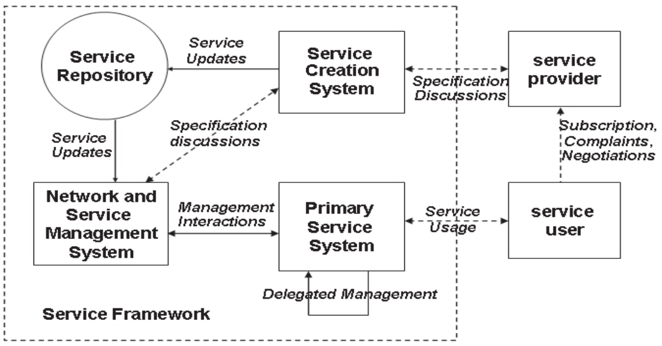


Figure 1: Adaptable service system life cycle.

The life cycle of adaptable service systems is depicted in Figure 1. *A service framework* is defined as a system for the specification, management, and execution of adaptable service systems. The service framework functionality is constituted by the *Primary Service System* itself, the *Service Creation System*, the *Service Repository*, and the *Network and Service Management System*. The *environment* consists of the administrative *service provider* and the *service user*.

The *Service Creation System* is concerned with the creation of executable specifications of the service system and its components according to the service provider's requirements. The *Primary Service System* provides services to the service users, while the *Network and Service Management System* provides management services to the Primary Service System. Both the Primary Service System and the Network and Service Management System are service systems according to the definition in Section 1.1.1. The boundary between these systems highly depends on the nature of the primary service, the nature of the service management functionality, and how the various functions are realised in software components. Some management functionalities can be integrated in the software components executing the primary service functionality. In Figure 1, this is indicated by the loop denoted as delegated management.

The functionalities of the service framework entities are here grouped in the functionality groups: *Service Creation*, *Node Configuration*, *Service Configuration*, *Service Provisioning*, and *Monitoring and Diagnosis*, as illustrated in Figure 2. These groups are not absolute, and the grouping can be done in several ways. The functions in the functionality groups are not necessarily single functions, and can in some cases be considered as functionality subgroups. With reference to Figure 1, Service Creation is carried out by the Service Creation System. The rest is done *in cooperation* between the Network and Service Management System and the Primary Service System. The following subsection explains the functions in each functionality group.

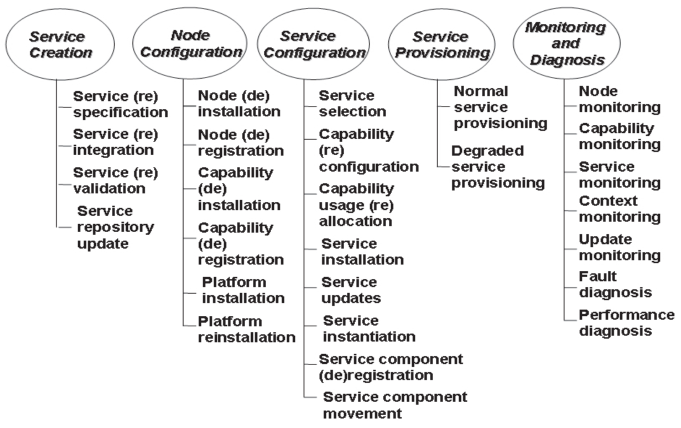


Figure 2: Adaptable Service System Life cycle Functionalities.

- **Service Creation:** *Service specification* is the process that designs and creates executable specification of service components constituting a service system. In the case of web service based functionality, *Service integration* can also comprise of service discovery, which is a process that involves searching and selecting service components that satisfy functional and performance requirements before integrating this functionality into a more comprehensive service system. *Service validation* is the process to confirm that the service component specification that will be deployed satisfies the requirements. Accordingly, the *Service Repository* will be *updated* with new version of the service component specification.

- **Node Configuration:** *Installation* is the provision of the physical existence of nodes and capabilities, while *registration* is the logical registration of nodes and capability instances. The physical installation is manually carried out by humans, while the logical registration can be done by a service system. However, this functionality will normally require cooperation with *a monitoring function* that can detect the physical presence of the component. *Platform reinstallation* is in general the installation after a node failure, but can also refer to the installation of a new version of the platform software.

- **Service Configuration:** *Service selection* is primarily a function used in cases where a change of context requires a new service. *Capability configuration* includes *Capability parameter configuration* as well as *Node selection*. *Capability parameter configuration* is the validation and the settings of capability parameter values according to a capability parameter configuration specification. *Node selection* is the selection of a node with respect to the required capability functionality and performance defined for the service component. *Capability usage allocation* determines the usage of allocated capabilities. *Service installation* is the deployment of the specifications of service components constituting a service system. Here, *Service update* is denoted as a separate function. This is because an update does *not* comprise of a new and complete installation of the specifications of *all* service components constituting a service system. *Instantiation* begins the execution of the installed or updated service component specifications. *Service component (de)registration* is the (de)registration of the service component instances in a service system, which can provide an overview of the instantiated service component instances. *Capability reconfiguration* can in general comprise of the movement of service components. *Service component movement* is different from installation as it includes the movement of an instance of a service component with present states and local variable values.

- **Service Provisioning:** *Normal service provisioning* refers to the offer of services with ordinary system performance. Here, *Degraded service provisioning* refers to the offer of services with degraded system performance; this requires adaptation actions. With respect to Figure 2, adaptation actions can incorporate *Capability usage reallocation* only, but in more serious cases, it can comprise of *Capability reconfiguration*, *Capability usage reallocation*, *Service component movement*, *Service component instantiation*, *Service component deregistration*, and *Service component registration*.

- **Monitoring and diagnosis:** *Node monitoring* monitors the existence and liveliness of nodes. *Capability monitoring* monitors the existence of capability types and the capability parameter values. *Service monitoring* similarly monitors the existence, liveliness, functionality, and performance parameter values of service components. *Update monitoring* monitors the existence of service component specification updates in the Service Repository. *Fault diagnosis* detects failures related to nodes, capabilities, and service components, while *Performance diagnosis* detects mismatches between the required system performance and the inherent system performance. Here, *Context monitoring* is related to the *adaptability terrain*, i.e., the environment in which the adaptable service system operates. The terrain can be classified as *physical* or *logical*. *Physical terrain* is defined by physical positions as well as the state of the terrain. Physical positions can be either the absolute position of the node in which the adaptable service system is executing or the position relative to other nodes and objects. State measures can be terrain type, temperature, concentration of gases, friction, and fluidity. *Logical terrain* is defined by logical positions and application layer associations between service components. An example of an adaptable service system is a robot snake [Lil11] that operates in a house on fire, which changes the movement types according to the terrain type and obstructed paths. Offered service examples include measurements, searching humans, video recording, and spraying of water. Context can also be defined to include capabilities, such as memory, CPU, battery, bandwidth, user preferences, and user profiles. However, in this thesis, capability is *not* considered as a part of the context. User preferences and profiles are considered as data, and are not visible in the presented functionality models.

1.1.3 Adaptability types

In this thesis, *adaptability types* are classified as *capability-related adaptation*, *functionality-related adaptation*, and *context-related adaptation*. *Capability-related adaptation* is defined as the ability to adapt because of a shortage of capabilities with appropriate logical functionality or an overload or failure of capabilities leading to reduced system performance. *Functionality-related adaptation* is the ability to adapt to new functionality requirements, and *context-related adaptation* is the ability to adapt to changes in the context. Combinations of these adaptability types are possible. However, only the capability-related adaptation is considered in this thesis.

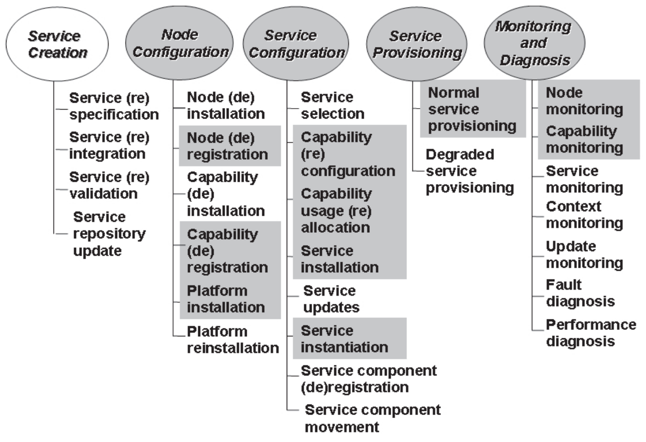


Figure 3: Basic Capability Awareness.

Capabilities are the basic foundation for the implementation of service systems. Therefore, a property required for any adaptability type is *to be aware of nodes and capabilities* and the ability to *configure service systems* according to the present availability of nodes and capabilities. This is denoted as *basic capability awareness.* The required functionalities are illustrated in Figure 3. In this case, it is assumed that there are no failures or overload. The functionalities required for the various adaptability types are defined as follows:

- **Capability-related adaptation:** This adaptability type requires functionality as illustrated in Figure 4. Here, the functionality needed for basic capability awareness is illustrated in grey, and the required additional functionality is shown in blue. In addition to the basic capability awareness, Platform (re)installation, Service component (de)registration, Service component movement, Degraged service provisioning, Service monitoring, Fault diagnosis, and Performance diagnosis is needed.

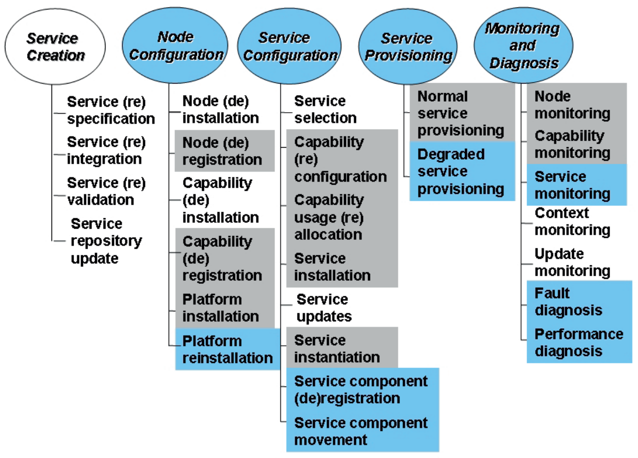


Figure 4: Capability-related Adaptation.

- **Functionality-related adaptation:** In addition to the functionalities of capability-related adaptation, the following functionalities are needed: Service (re)specification, Service (re)integration, Service (re)validaton, Service repository update, Update monitoring, and Service update. Fault and Performmance diagnosis are included for covering the cases where faults or performance lead to the redesign of the service system specification and platform software. Degraded service provisioning covers cases where new specification requires Capability reconfiguration because of reduced system performance. Service creation functionalities require human involvement. The functionalities that can be automated, however, are Service repository update and Service update [DDP11].

- **Context-related adaptation:** In addition to the basic capability awareness, the following functionalities are needed: Service monitoring, Context monitoring, and Service selection. Context-related adaptation is related to the adaptability terrain. Context, as defined by the physical position, is used in a wide variety of commercial user services such as ticket sales, visiting touristic places, and restaurant services [GER11]. Context-related adaptation is to some extent pre-programmed. Physical terrain adaptation can be programmed mathematically. Logical terrain adaptation is realised by client applications, i.e., client service components that are flexible and able to work with new server service components. In addition, user interaction is needed to make a selection among the available applications.

1.1.4 Closed feedback loop

The adaptability can be realised by a *closed feedback loop*, which is a mechanism originally based on the control theory [DHP05] that deals with dynamical behaviours of a system. Figure 5 depicts the basic entities in a closed feedback loop system. The objective of a closed feedback loop is to determine appropriate solutions from the *controller* according to the *system-desired output*. The solutions are to dynamically change the *controlled system* behaviour, which results in the desired *system output*. These solutions can be considered as *adaptation plans*. An adaptation plan consists of one or more *adaptation actions*. The *feedback* is the information obtained from measuring the system output after the adaptation plan has been applied, and is fed to the controller to establish the adaptation actions for the next cycle.

In some cases, the feedback is also used to *"adapt"* the adaptation action selection as necessary. For these cases, the consequence of the adaptation actions can be derived from the feedback, which will affect the adaptation action selection. The adaptation of the selection can include changing the algorithms used to select the adaptation actions, and/or changing the specifications of the adaptation actions. In other cases, however, the feedback does not affect the adaptation action selection. The adaptation action selection is *"repeated"* through the same algorithms and the same adaptation action specification as in the previous cycles.

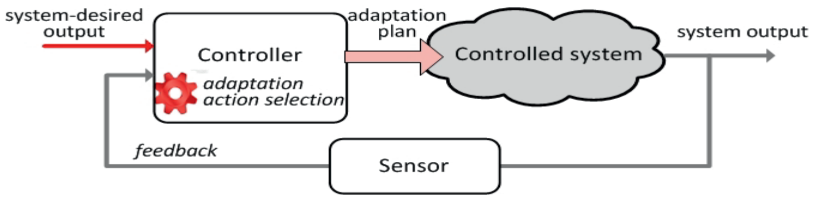


Figure 5: Basic Closed Feedback Loop System.

For the adaptable service systems defined in this thesis, the system-desired output includes the desired system performance and service income; accordingly, the feedback consists of the inherent system performance and the service income observed during an interval. The adaptation actions are, for example, to change states of the service components and to initiate or terminate the service components.

In this thesis, the systems implementing the closed feedback loop are classified into five types (Type A–E) as follows:

A) Pure algorithm system (without policy and learning): This system uses only algorithms for generating the adaptation plan. That is, the adaptation actions are hard coded as a part of the controller's algorithm. In some cases, the system-desired output is also hard coded in addition to the adaptation actions. However, as compared to Type B–E below, this system is not flexible. It can provide adaptability in terms of the adaptation plan only for a set of previously known problems. An example of this type of closed feedback loop system is found in Paper D.

B) Policy-based system (without learning): In this system, *policies* are used to specify the adaptation actions. The policies are assigned to the controller as input, in addition to the feedback and the system-desired output. The controller determines the appropriate adaptation actions based on these policies. These policies can formally be defined by an information model. In the same way, the system-desired output can be defined by an information model. This type of closed feedback loop system provides flexibility, because the adaptation action specification (i.e., the policies) as well as the system-desired output can be changed, added, and removed by the system engineer during the system runtime. An example of this is found in Paper E.

C) Policy-based ontology system (without learning): This closed feedback loop system is a subtype of Type B. The policy is defined by an ontology and is represented by a language for ontologies. An ontology is a formal and explicit form of a shared conceptualisation [SBF98]. An example of this is found in [HDS08].

D) Policy-based system with learning: This system uses policies for the adaptation action specification, as in Type B. In addition, however, the system has a *learning* mechanism that uses the feedback to adapt the adaptation action selection.

As explained above, the learning mechanism can adapt the selection in three ways: i) by changing the algorithms used to select the adaptation actions, ii) by changing the specification for the adaptation actions (i.e., the policies), and iii) by changing both the algorithms and the policies. This means that the algorithms and/or policies of this system can be changed, created, and removed by the system itself during the system runtime. An example of this type of closed feedback loop system is found in Paper E, where only case ii) is implemented.

In this thesis, these changeable policies are denoted as *dynamic policies*. The dynamic policy changes in cases ii) and iii) above can be on *entities of the policies*, on the *policy priorities*, and on the *policy usage statuses* (e.g. active, inactive, and deleted). An example of the use of the policy priorities is as follows: "When there is a set of defined dynamic policies, only some of them – those which have the highest priority – will be considered by the controller. After the adaptation actions as defined by a policy have been applied, the priority of this policy will be changed based on the feedback. Therefore, the dynamic policies that are considered by the controller in each cycle are different."

E) Policy-based ontology system with learning: This closed feedback loop system is a subtype of Type D. The policy is defined by an ontology and is represented by a language for ontology. Examples of this type are found in Paper F and [SRS07].

The service framework contributions presented in Section 4.1 can be used to implement the closed feedback loop systems for all types. The relationship between these contributions and the closed feedback loop systems is discussed in Section 4.2.

1.2 Outline of the thesis

The thesis is structured into two main parts: Part I – Overview, and Part II – Research papers. Part I is furthered structured as follows: *Section 1* presents an introduction of the thesis, including general issues of adaptable service systems, an outline of the thesis, and a guideline for reading Part II. *Section 2* presents the research method. This includes the objectives, scope, problem statements, and the research cycle of the work. *Section 3* presents an overview of TAPAS concepts, architectures, and service execution platforms. *Section 4* presents the service framework, the thesis contributions, the realisation of the closed feedback loop systems, and finally the realisation of the problem statements as defined in Section 2. *Section 5* gives a summary of the papers included in Part II. *Section 6* describes related works, and *Section 7* presents the conclusions and further work.

The papers included in Part II are as follows:

Paper A	Ontology-based Capability Self-Configuration P. Thongtra, F.A. Aagesen, and N. Sriharee, in *Proc. 14th Eunice Open European Summer School on which networks for which services*, Brest, France, September 2008.
Paper B	Capability Ontology in Adaptable Service System Framework P. Thongtra and F.A. Aagesen, in *Proc. IARIA/IEEE 5th Int. Multi-Conf. Computing in the Global Information Technology (ICCGI 2010)*, Valencia, Spain, September 2010.
Paper C	OWL-based Node Capability Parameter Configuration P. Thongtra, F.A. Aagesen, and K. Dittawit, in *Proc. 18th EUNICE/IFIP WG 6.2, 6.6 Int. Conf. Information and Communications Technologies*, Budapest, Hungary, August 2012. *Lecture Notes in Computer Science (LNCS) 7479*, pp. 124-135, 2012.
Paper D	An Adaptable Capability Monitoring System P. Thongtra and F.A. Aagesen, in *Proc. IARIA/IEEE 6th Int. Conf. Networking and Services (ICNS 2010)*, Cancun, Mexico, March 2010.
Paper E	Towards Policy-Supported Adaptable Service Systems P. Supadulchai, F.A. Aagesen, and P. Thongtra, in *Proc. 13th Eunice Open European Summer School and IFIP TC6.6 Workshop on*

Dependable and Adaptable Networks and Services, Twente, Netherlands, June 2007. *Lecture Notes in Computer Science (LNCS) 4606*, pp. 128-140, 2007.

Paper F An Autonomic Framework for Service Configuration
P. Thongtra and F.A. Aagesen, in *Proc. IARIA/IEEE 6th Int. Multi-Conf. Computing in the Global Information Technology (ICCGI 2011)*, Luxembourg, June 2011.

Paper G On Capability-related Adaptation in Networked Service Systems
F.A. Aagesen and P. Thongtra, *Int. Journal of Computer Networks & Communications (IJCNC)*, vol. 4, no. 4, July 2012.

1.3 Guideline for reading Part II

The papers included in Part II are self-contained. Since these papers have some overlapping content and can be linked, it is proposed that the reader follows the suggested reading order as illustrated in Figure 6.

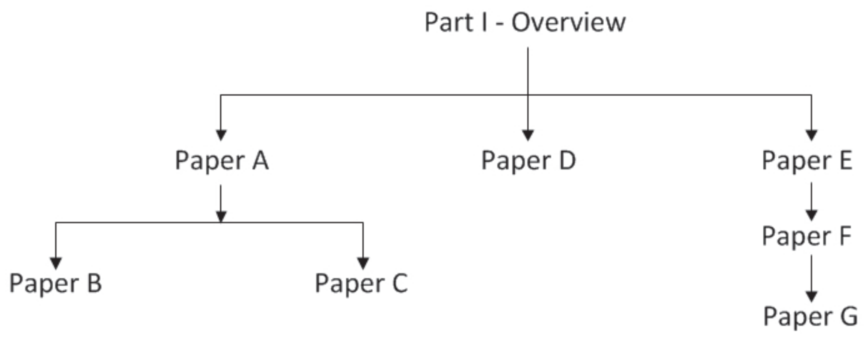


Figure 6: Suggested paper reading order.

From the figure, Part I is followed by Paper A, D, or E. After finish Paper A, the reader can continue reading Paper B or C. Paper E is followed sequentially by Papers F and G.

2 Research method

This section describes the research objectives, scope, problem statements, and a research cycle used for researching. Section 2.1 presents the overall research objectives. Section 2.2 describes the scope of the research, Section 2.3 illustrates the problem statements, and the research cycle is stated in Section 2.4.

2.1 Research objectives

The research objectives are "*to specify, construct, validate, and evaluate a service framework for capability-related adaptation in adaptable service systems.*" The service framework consists of a computing architecture, concepts, ontologies, service component models, mechanisms, system architectures, and service execution platforms.

2.2 Scope

Security and dependability aspects should be included in real-world adaptable service systems; however, such considerations are out of the scope of this thesis. It is possible that some of the primary service functionalities and management functionalities can be distributed among several service components in a decentralised architecture. However, the dependability is not the main focus of this architecture design. Here, the architectures presented are not dependable, and the necessary replication of nodes, capabilities, and functionalities is not properly handled.

2.3 Problem statements

The problem and sub-problem statements for this thesis are defined as follows:

- **P1**: Which architecture can be used for the specification and execution of service components in adaptable service systems? Which concepts are needed, and how can these concepts be included in the architectural framework?

- **P2**: How can the concepts be formally defined? Can ontologies be used? How can we model and represent the concepts in these ontologies?

- **P3**: Which types of service components, mechanisms, and features can be used to implement the functionalities required for capability-related adaptation with the consideration of flexibility and effectiveness?

- **P4**: Concerning the autonomic service systems:
  - What is an Autonomic Element model? How can we specify the Autonomic Element functionality? Can EFSM and policy specification be used?
  - Which learning mechanisms can be used?
  - Can Autonomic Elements implement the functionalities required for the capability-related adaptation?

- **P5**: How can we validate and evaluate the proposed framework?

2.4 Research cycle

The research cycle used in this thesis is based on the common scientific iterative research cycle method. The research cycle consists of iterations of the following phases:

Problem formulation

A set of problem statements is identified in this phase. Such problem statements usually arise as a result from a literature review, which evaluates and analyses the current state of the field in question.

Solution proposition, modelling, and formalisation

Here, a service framework is proposed for the problem statements. The service framework is formalised to ensure integrity, soundness, consistency, and completeness. Using formalised approaches is easier in order to further improve and add new features, as opposed to non-formalised ones.

Validation and evaluation

Demonstration and simulation is then applied for validating, evaluating, and improving the proposed solution. Demonstration is a method for illustrating how the proposed service framework performs in a *real environment* by using *prototypes*. Simulation is a method for testing the proposed service framework in a *setup environment* by using *simulation software*. The demonstration method is used when the system performance evaluation is not the main focus. Simulation is typically used when real-environment demonstrations are not practical, and when the applications of the system models are needed in order to give more detailed results with respect to the system performance.

For the demonstration method, prototypes are implemented based on the proposed service execution platforms in this thesis. The prototype of capability parameter configuration systems and the prototype of adaptable monitoring systems have been implemented and presented in Papers C and D, respectively.

In terms of the simulation method, simulation software can be flexibly implemented by any simulation tool or programming language. The simulation software developed with the Java programming language is used in Papers B, E, F, and G.

Results discussion

Here, results from the validation and evaluation are discussed, and conclusions are accordingly drawn. If the results are not as expected, the solution will be improved and/or some assumptions will be changed. This may subsequently lead to a new problem statement or a new service framework and thus begin a new iterative research sequence.

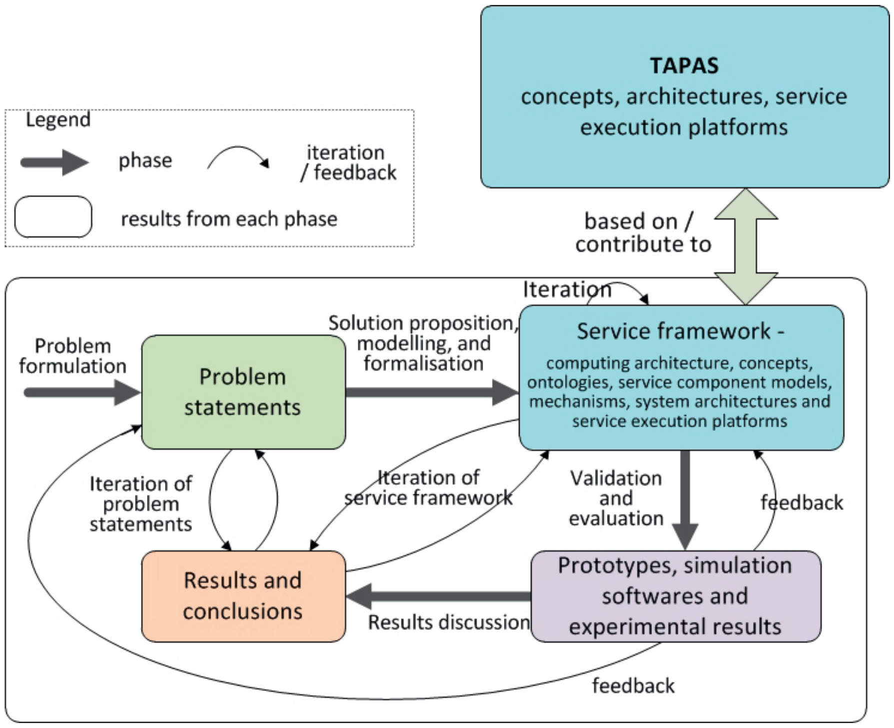

TAPAS
concepts, architectures, service execution platforms
Legend
phase
iteration / feedback
results from each phase
based on / contribute to
Iteration
Problem formulation
Problem statements
Solution proposition, modelling, and formalisation
Service framework - computing architecture, concepts, ontologies, service component models, mechanisms, system architectures and service execution platforms
Iteration of problem statements
Iteration of service framework
Validation and evaluation
feedback
Results and conclusions
Prototypes, simulation softwares and experimental results
Results discussion
feedback

Figure 7: Research cycle.

Figure 7 illustrates an overview of the research cycle used in this thesis, which consists of problem formulation, solution proposition, modelling and formalisation, validation and evaluation, and results discussion. This figure also shows that the proposed service framework is both based on and contributes to TAPAS.

3 TAPAS

The TAPAS (Telematics Architecture for Play-based Adaptable Service Systems) project aims at developing concepts, architectures, service execution platforms, and prototype implementations for adaptable service systems. The adaptability types supported by TAPAS are capability-related adaptation, functionality-related adaptation, and context-related adaptation, as defined in Section 1.1.3.

The service framework presented in this thesis is related to TAPAS. On one hand, the TAPAS concepts, architectures, and service execution platform that existed at the starting point of this Ph.D. study are in fact the basis of this thesis. This foundation had been built by several Ph.D. theses [Shi05][Jia08][Sup08] and Master's theses [Lüh04][Vil04][Nis08]. On the other hand, the research work presented in this thesis has also contributed to new TAPAS concepts, ontologies, service component models, mechanisms, system architectures, and service execution platforms.

This section gives an overview of TAPAS. Section 3.1 presents architectures, including their concepts and features. Section 3.2 introduces TAPAS service execution platforms, and the research work contributed to TAPAS is summarised in Section 3.3.

3.1 TAPAS architectures

The TINA two-dimensional architectures [BDD99] were followed. TAPAS architecture is separated into a *computing architecture* and a *service functionality architecture*. The *service functionality architecture* is focussed on the structural arrangement of service functionalities that are independent of implementation. The *computing architecture* focuses on the modelling of functionality with respect to implementation, but is independent of the nature of the service functionality. The computing architecture is composed of concepts for the specification and execution of service components; thus, it is the basis for the creation of service systems.

3.1.1 TAPAS computing architecture

The computing architecture is founded on a *theatre metaphor*, where *actor*, *manuscript*, *role figure*, *play*, and *capability* are core concepts. *Actors* perform *roles* according to *manuscripts*. Here, the actors are generic operating system software components in the nodes that can download manuscripts to be played. An actor who is behaving according to a manuscript is denoted as a *role figure*. A *play* consists of several *actors* playing different *roles*, each possibly having different requirements on *capabilities*.

Figure 11 in Section 4.1 illustrates the present version of the computing architecture. The computing architecture has three views: the *service view*, *play view*, and *physical view*. In the service view, the concept service system, service component, service functionality, and service performance are defined as in Section 1.1.1. *SLAs* are agreements between the service users and the service provider. SLA classes can contain elements such as required service functionalities and performance, payment for the service when the agreed service performance is offered, and penalty in case of reduced service performance. *Service income* is the sum of the payment income and the penalty cost. *Inherent service income* is defined as the service income observed during an interval.

Leaf service components that cannot be further decomposed consist of *role figures*. A role figure can have dialogues with other role figures. A *director* is an actor with supervisory functions with regard to other actors. A play consisting of role figures and dialogues is a service system as seen from the play view.

In the physical view, the core platform is a service execution platform that supports the execution of service functionality based on the computing architecture functionality. The core platform will be explained more fully in Section 3.2.

3.1.2 TAPAS service functionality architecture

The service functionality architecture consists of *primary service functionalities* and *management functionality components* as illustrated in Figure 8. Here, five repositories are defined: Service specification repository (SpcRep), Capability type repository (CapRep), Inherent capability and service repository (InhRep), Context repository (ConRep), and Platform repository (PltRep). SpcRep stores SLAs, the service components' behaviour specifications, as well as capability functionality and performance requirements. These service component specifications are EFSM, policy, and goal specifications. CapRep stores the capability concepts and the capability parameter configuration specifications. InhRep stores data about available nodes, capability instances, and instantiated service components. ConRep stores predefined context states, while PltRep stores the platform software needed to execute service functionality.

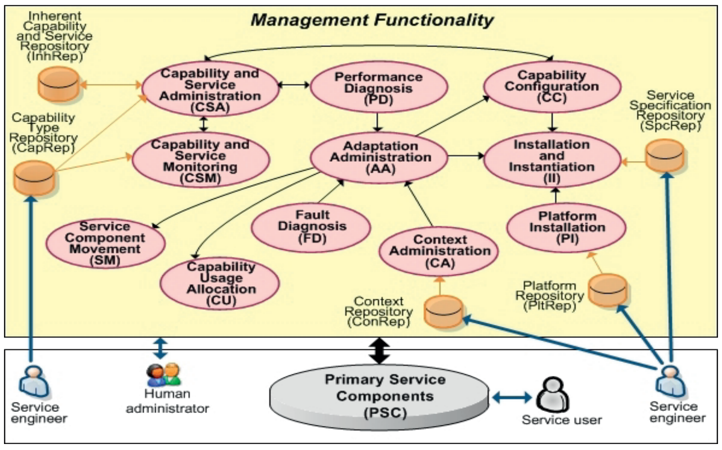

Management Functionality
Inherent Capability and Service Repository (InhRep)
Capability Type Repository (CapRep)
Capability and Service Administration (CSA)
Performance Diagnosis (PD)
Capability Configuration (CC)
Service Specification Repository (SpcRep)
Capability and Service Monitoring (CSM)
Adaptation Administration (AA)
Installation and Instantiation (II)
Service Component Movement (SM)
Fault Diagnosis (FD)
Context Administration (CA)
Platform Installation (PI)
Capability Usage Allocation (CU)
Context Repository (ConRep)
Platform Repository (PltRep)
Service engineer
Human administrator
Primary Service Components (PSC)
Service user
Service engineer

Figure 8: Service Functionality Architecture.

The functionality components illustrated in Figure 8 implement functions for capability-related, functionality-related, and context-related adaptation. The functionality components *Capability configuration (CC)*, *Capability usage allocation (CU)*, *Service component movement (SM)*, *Platform installation (PI)*, *Fault diagnosis (FD)*, and *Performance diagnosis (PD)* correspond directly to the functions defined in Section 1.1.2. *Capability and service administration (CSA)* performs Node (de)registration, Capability (de)registration and Service component (de)registration. A view of InhRep is also provided. *Capability and service monitoring (CSM)* performs node, capability, and service monitoring. The result of the monitoring is then reported to CSA. *Capability configuration (CC)* generates configuration plans for service components, while a configuration plan defines the nodes for deployment and instantiation. *Capability usage allocation (CU)* allocates capabilities in accordance with the present system performance, SLAs, and the optimisation criteria chosen by the service provider. *Installation and instantiation (II)* consists of the execution of the configuration plan as well as the monitoring and installation of new service component specifications. The configuration plan execution results in the deployment and instantiation of service component specifications in the selected nodes. *Service component movement (SM)* manages the ongoing sessions on behalf of a moving service component during movement. *Adaptation administration (AA)* plans and administers a reconfiguration initiated by unwanted events or states during the normal service system execution. This can be initiated by FD, PD, CA, or by the human administrator. The reconfiguration can result in CU only, or the combination of CC and CU, including SM and II. *Context administration (CA)* monitors contexts and initiates the configuration of context-dependent application software. *Platform installation (PI)* is bootstrap functionality that installs needed platform software when nodes are (re)started and when there is a new version of the platform software. *Primary service components (PSC)* essentially implement the service provisioning.

There is, however, not always a clear boundary between primary service functionalities and management functionalities. Most primary service systems need capabilities and functionality components such as *PD*, *CU*, and *CC.* Such functionalities can often be designed as a part of the primary service systems.

With respect to the service functionality architecture, *the functionality components CSA, CSM, CC, CU, SM, PI, FD, PD and PSC*, as well as *a part of AA and II*, *correspond to the capability-related adaptation functionalities*; these are considered in this thesis.

3.2 TAPAS service execution platforms

TAPAS service execution platforms are software frameworks for the deployment, execution, and management of service systems that are defined by the TAPAS concepts and architectures. The platforms are divided into a *core platform* and a *management platform*. The core platform supports the execution of service functionality based on the aforementioned computing architecture functionality. The management platform supports the management functionalities as defined by the service functionality architecture, e.g. service component movement.

The core platform is essentially an actor of instantiation functionality, a manuscript for execution functionality, and constitutes a basic communication functionality between actors. Both platforms are implemented based on the Java programming language. Here, a set of message and procedure APIs to be used by the service engineers are defined. Java socket [Lüh04][Nis08], Java RMI [Aag01], and XML Web services [Vil04] can be used as the communication mechanisms between the actors.

3.3 Thesis contributions to TAPAS

At the start of this Ph.D. study, only a part of the functionalities required for capability-related adaptation, functionality-related adaptation, and context-related adaptation had been considered. A comprehensive study on mobility management can be found in Mazen Malek Shiaa's thesis [Shi05]. The capability configuration management, which covers the functionality components CU, PD, PSC, and parts of CSA, CSM, CC, and AA, is presented in Paramai Supadulchai's thesis [Sup08]. In Shanshan Jiang's thesis [Jia08], service specification and integration are studied.

The service framework presented in this thesis is mainly based on the thesis of Paramai Supadulchai – Reasoning-based Capability Configuration Management in Adaptable Service Systems [Sup08]. However, this thesis contributes new concepts, ontologies, service component models, mechanisms, system architectures, and service execution platforms in order to fulfil the complete support of the capability-related adaptation. These contributions will be explained in detail in Section 4. In this subsection, a brief comparison of the TAPAS architectures and service execution platforms at the initial stage of this Ph.D. study as compared to the present is given. Figure 9 illustrates these differences.

Concerning the computing architecture, new concepts such as Learning Mechanism (LM) and goal specification have been added. In addition, the concept capabilities, Reasoning Machine (RM), EFSM specification, and policy specification have been extended. The capability types and parameters are defined by the capability ontology. A new manuscript defined by a combination of two EFSM specifications, a policy, and a goal specification, as well as Autonomic Element (AE)-based role figures executing this manuscript are introduced. In addition, generic states of role figures are defined.

Concerning the service functionality architecture, the management functionalities have been revised and extended. The management and primary service functionalities in the current version of this architecture enable capability-related adaptation, functionality-related adaptation, and context-related adaptation, as mentioned above. The functionality components PI, FD, and CA have been added, while the functionalities of CSA, CSM, CC, AA, and II have been revised and extended. In addition, the policy-based reasoning has been extended. The policy specification can be based on both the goal-based policy ontology and CIM. Action rewarding, which is a learning mechanism, is introduced. Both the policy-based reasoning and action rewarding are support functionalities required for both the primary service and management functionalities.

In terms of the service execution platforms, the core platform has been extended. The current version can support the concurrent execution of *several* service systems in a set of available nodes. The core platform has been improved to incorporate the ability to interpret the capability functionality and performance requirements that are expressed in OWL and OWL/XDD. The management platform is new and has been implemented. The current version of the management platform supports the functionality components SM and CC, as well as parts of CSA and CSM. The platform software has been uploaded to a shared web server, and thus is ready to be dynamically downloaded by the bootstrap program installed in the nodes. Three technical documents about TAPAS service execution platforms, including TAPAS platform messages and procedures,

TAPAS platform execution framework, and a service system implementation guideline, were written. They are available at
http://tapas.item.ntnu.no/wiki/index.php/TAPAS_Platform.

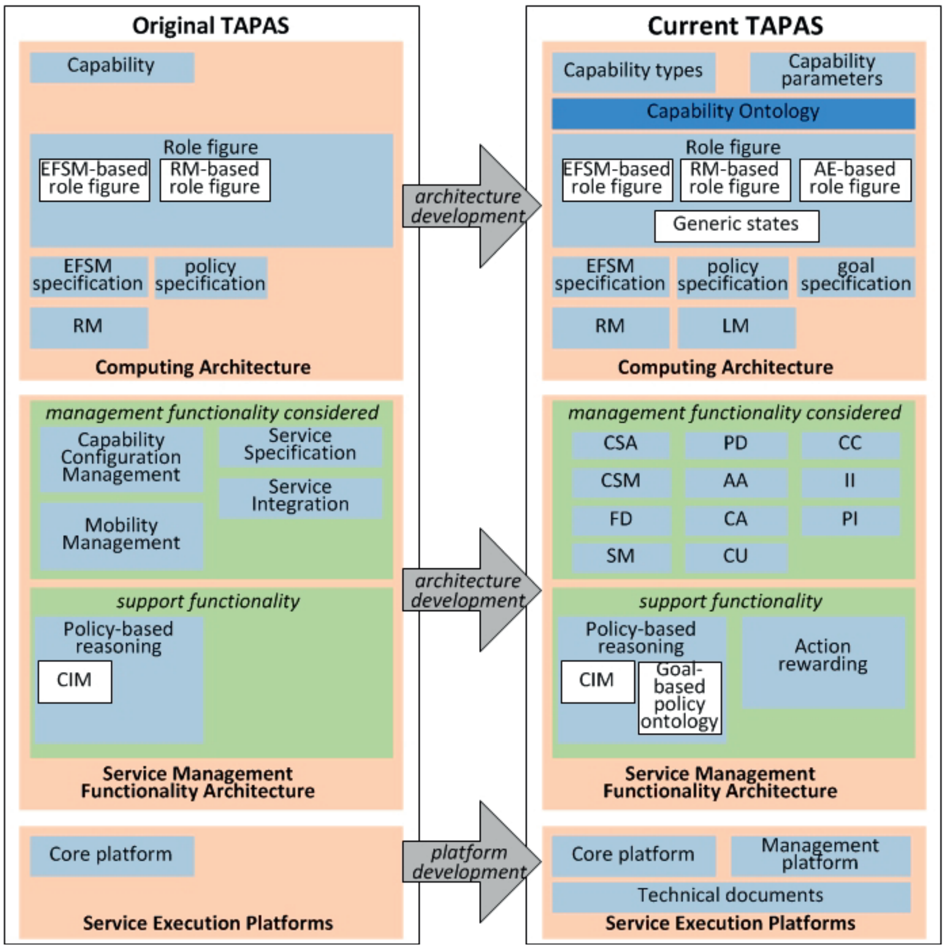


Figure 9: Thesis contributions to TAPAS.

4 The service framework

The service framework for capability-related adaptation is defined to consist of six contributions (**C1–C6**), as follows:

C1: Ontology-based computing architecture

C2: Ontologies, models, and representations

C3: Policy-based reasoning

C4: Autonomic-based model

C5: Capability-related adaptation functionality realisation

C6: Service execution platforms, prototypes, and simulations

Each contribution consists of sub-contributions which represent contributed computing architecture, concepts, ontologies, service component models, mechanisms, system architectures, and service execution platforms. These contributions and sub-contributions are summarised and concluded in Section 4.1. However, they are presented in much more detail in the papers of Part II.

The remainder of this chapter proceeds as follows. Section 4.1 presents the contributions and sub-contributions. Section 4.2 discusses the relationships between the contributions and the closed feedback loop systems as defined in Section 1.1.4. The relationships between the contributions and the problem statements as defined in Section 2.3 are discussed in Section 4.3. This section also describes the relationships between the contributions and the papers in Part II.

4.1 The service framework contributions

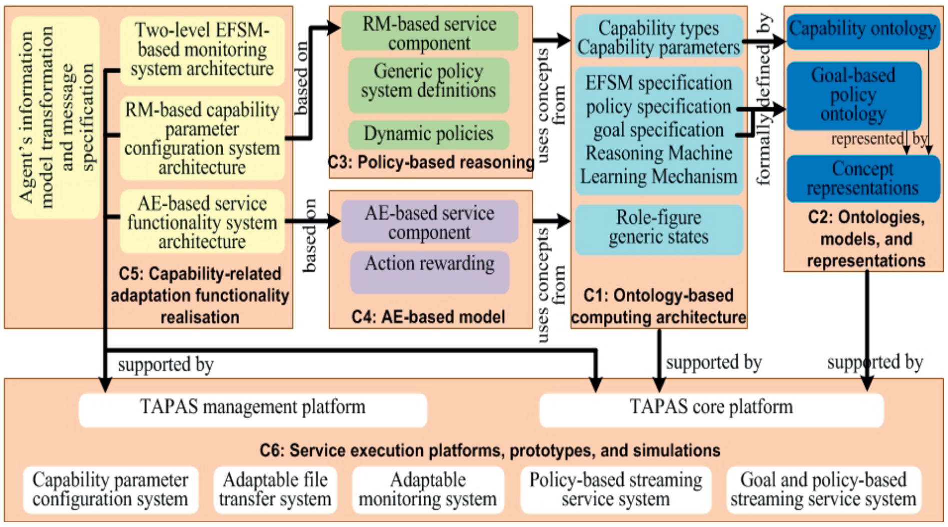

Agent's information model transformation and message specification
Two-level EFSM-based monitoring system architecture
RM-based capability parameter configuration system architecture
AE-based service functionality system architecture
C5: Capability-related adaptation functionality realisation
based on
RM-based service component
Generic policy system definitions
Dynamic policies
C3: Policy-based reasoning
based on
AE-based service component
Action rewarding
C4: AE-based model
uses concepts from
uses concepts from
Capability types
Capability parameters
EFSM specification
policy specification
goal specification
Reasoning Machine
Learning Mechanism
Role-figure generic states
C1: Ontology-based computing architecture
formally defined by
Capability ontology
Goal-based policy ontology
represented by
Concept representations
C2: Ontologies, models, and representations
supported by
supported by
supported by
TAPAS management platform
TAPAS core platform
C6: Service execution platforms, prototypes, and simulations
Capability parameter configuration system
Adaptable file transfer system
Adaptable monitoring system
Policy-based streaming service system
Goal and policy-based streaming service system

Figure 10: Service framework for the capability-related adaptation.

The service framework contributions are related to each other. Figure 10 illustrates these contributions, sub-contributions, and their relationships. The ontology-based computing architecture in **C1** provides concepts for the service component specification and execution. Some of these concepts are further defined by the capability and goal-based policy ontologies in **C2**. The RM-based service component, including the policy-based reasoning functionality, is provided in **C3**. The AE-based service component and its functionalities are given in **C4**. The system architectures in **C5** define specific functionalities, interworkings, and structural organisation of the EFSM-, RM- and AE-based service components, which implement the functionalities required for capability-related adaptation. The service execution platforms in **C6** can both i) support the execution of service functionality based on the ontology-based computing architecture functionality, and ii) provide management support for the service systems.

C1: Ontology-based computing architecture

This contribution relates to concepts and features in the *ontology-based computing architecture*, which expands upon the previous version in TAPAS. The contribution consists of *capability types*, *capability parameters*, *EFSM specification*, *policy specification*, *goal specification*, *the Reasoning Machine*, *the Learning Mechanism*, and *role figure generic states*. Figure 11 illustrates the ontology-based computing architecture. The new and revised concepts are in blue.

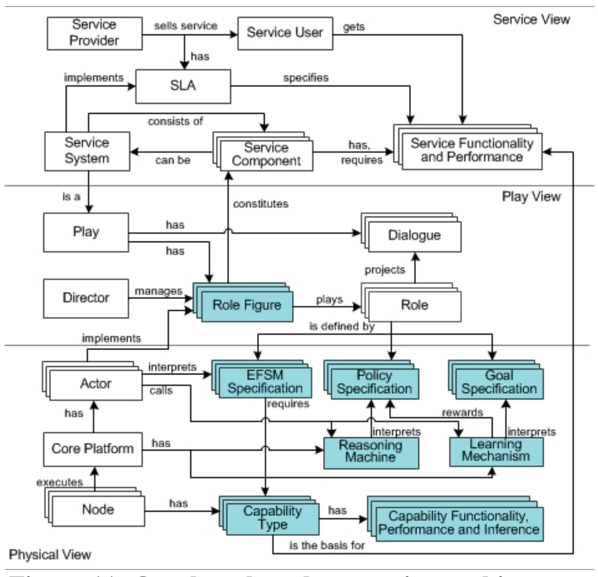


Figure 11: Ontology-based computing architecture.

Capability types and capability parameters

The capability types and capability parameters, as defined in Section 1.1.1, have been revised and extended. As the capabilities are the basis for the implementation of the service systems, these capability concepts can be found in all the papers of Part II.

The capability types and parameters, as well as other necessary capability-related concepts that are not visible in the ontology-based computing architecture, are further defined by the capability ontology. This will be presented below in **C2**.

EFSM, policy and goal specification, Reasoning Machine, and Learning Mechanism

The manuscripts introduced in Section 3.1.1 are the behaviour specifications of roles. In the ontology-based computing architecture, the manuscripts can be defined by:

i) *An EFSM specification*,
ii) *A combination of an EFSM and a policy specification*, or
iii) *A combination of two EFSM specifications, a policy, and a goal specification*.

An actor can execute the EFSM specification. The policy and goal specifications are executed by the Reasoning Machine (RM) and the Learning Mechanism (LM), respectively. Accordingly, **i)** an actor behaving according to the manuscript is denoted as an *EFSM-based role figure*, **ii)** an actor with RM behaving according to the manuscript is denoted as a *RM-based role figure*, and **iii)** an actor with RM and LM behaving according to the manuscript is denoted as an *AE-based role figure*.

An EFSM-based role figure (actor) that downloads and executes the EFSM specification is the original concept of TAPAS. However, the EFSM specification has been revised from its previous version to support service component mobility. Specifically, the stable state concept is explicitly included. A stable state is one in which a role figure can move safely; this means that its behaviours are suspended for a short period, and are reinstantiated at a new node with the present state, local variables, and the waiting received messages. The revised EFSM specification is presented explicitly in Papers D and F. However, EFSM-based role figures and EFSM specifications are used in all papers. XML is used for the representation of the EFSM specification, as carried out in [Jia08].

The RM-based role figure and policy specification will be presented below in **C3**, and the AE-based role figure and goal specification is given in **C4**.

Role figure generic states

The generic states of the role figure found in Papers F and G are {Initial, Normal, Degraded, Moving, Idle, Terminated}. These states indicate common statuses of role figures in any adaptable service systems. Based on these generic states, an overview of all instantiated service components constituting a service system can be provided.

C2: Ontologies, models, and representations

An ontology is a formal and explicit specification of a shared conceptualisation [SBF98]. *An ontology language* is a machine understandable language that expresses the ontology concepts as well as their relations and instances. There are two ontologies presented in this thesis: *capability ontology* and *goal-based policy ontology*. The ontology concept representations based on OWL [W3C04] and OWL/XDD [WY04] are also presented. The following subsection summarises the ontologies and their concept representations.

Ontologies are applied in this thesis because in general, an ontology is a *unified model* enabling the definitions of both *non-rule-based* and *rule-based concepts*. In addition, ontologies support *interoperability*, *aggregation*, and *powerful reasoning*. *Interoperability* means that an ontology's concepts applied by a service component in a service system can be shared and interoperated among other service components in various service systems. *Aggregation* means that an ontology can be integrated with other ontologies from different application domains. As an example, the capability ontology can be integrated with the service ontology [Jia08].

With respect to *reasoning*, it is a mechanism that derives new implicit information from explicit existing information based on mathematical logic. However, the reasoning for information defined by rule-based ontology concepts is more powerful than general reasoning for information defined by rules in other information models. This is because ontologies have comprehensive built-in constructors for the definition of rules, and accordingly, the definitions of complete and sound rule-based ontology concepts enable powerful reasoning. On the other hand, the reasoning based on other information models is also possible. For example, CIM [DMT12] is applied in Paper E. The reasoning based on capability ontology and goal-based policy ontology will be explained in more detail in **C3**.

Capability ontology

Capability ontology was initially presented in Paper A. The capability parameter concept was not explicitly defined in the initial version; thus, its usage was limited and was appropriate in only some application domains. The capability ontology has since been revised and extended, as found in Papers B and C. The current version of the capability ontology consists of a *node*, *capability types*, *capability parameters*, and *service management functions*. Figure 12 illustrates the capability ontology model. The node, capability types, and capability parameters are defined as above. A *service management function* is a capability-related action with constraints. The service management functions are a part of the capability ontology, in order to enable the interoperability of the service management function concept.

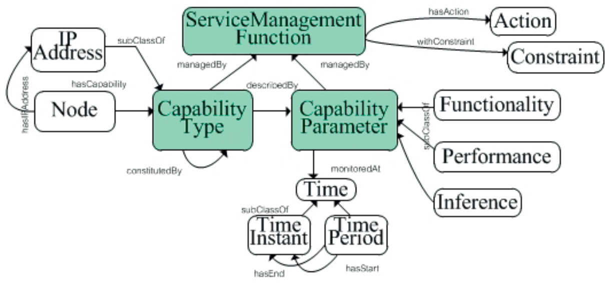


Figure 12: Capability ontology model.

The definitions of the ontology concepts can be found in Papers B and C. However, in Paper C, the capability ontology, types, and parameters are prefixed by "node" to prevent confusion between these and the NETCONF capability from the NETCONF network management framework [Enn06].

In general, the concepts of managed object information models for various network management frameworks (e.g. SMI/ASN.1 [Mcc99] for SNMP, CIM/XML [DMT12] for WBEM, and YANG/XML [Bjö10] for NETCONF) are a subset of the capability ontology concepts. Papers B and C present the case where the SNMP MIB managed objects are considered, and these are applied as the capability types and non-rule-based capability parameters.

Goal-based policy ontology

Goal-based policy ontology includes the concepts of *goals*, *policies*, and *inherent states*. Figure 13 illustrates the goal-based policy ontology model. The complete definitions of these ontology concepts are found in Papers F and G.

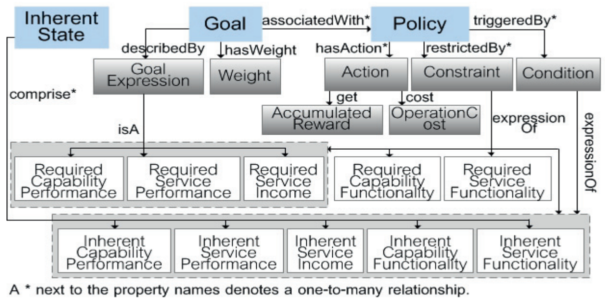

Inherent State
Goal
associatedWith*
Policy
triggeredBy*
describedBy
hasWeight
hasAction*
restrictedBy*
Goal Expression
Weight
Action
Constraint
Condition
comprise*
get
cost
Accumulated Reward
OperationCost
expression Of
expressionOf
isA
Required Capability Performance
Required Service Performance
Required Service Income
Required Capability Functionality
Required Service Functionality
Inherent Capability Performance
Inherent Service Performance
Inherent Service Income
Inherent Capability Functionality
Inherent Service Functionality
A * next to the property names denotes a one-to-many relationship.

Figure 13: Goal-based policy ontology model.

Policies are *rules* defined by *conditions*, *constraints*, and *actions*, which accordingly define the actions that adapt the service system in states with unwanted system performance and service income. A *constraint* restricts the usage of the policy. It is indicated by a Boolean expression, which consists of required and inherent functionality, performance of services and capabilities, required and inherent service income, as well as available nodes, their capabilities, and system time. A *condition* defines an inherent state in which the service system requires adaptation actions. A condition is similar to the constraint indicated by a Boolean expression, consisting of inherent functionality and performance of services and capabilities, as well as inherent service income. Thus, a *policy* fires when its *condition* and *constraint* expressions are true, which triggers the corresponding *actions* to be executed.

Concerning the *aggregation* defined above, the capability ontology concepts are integrated into the goal-based policy ontology. Some of the capability ontology concepts (e.g. the capability types), however, are invisible in the goal-based policy ontology model depicted in Figure 13. The service management functions are "*capability-related simplified policies*" that have *no conditions* and focus on the capabilities. That is, the constraints of these capability-related simplified policies are based on the required and inherent functionality and performance of capabilities, the required capability types, as well as the available nodes and their installed capabilities. The actions of these are to manage the capabilities, and to have *no accumulated rewards* and *operation costs*.

Concept representations

OWL (Web Ontology Language) [W3C04] and OWL/XDD (XML Declarative Description Language) [WY04] are used to represent the ontology concepts. OWL is a W3C standardised language for the representation of ontologies. OWL provides a rich set of constructors for the representation of *non-rule-based concepts*. OWL/XDD extends ordinary OWL elements by incorporating *variables* for the enhancement of expressive power and the representation of implicit information; thus, this is used to express *rule-based concepts*. Examples of the ontology concept representations in OWL and OWL/XDD are available at http://tapas.item.ntnu.no/wiki/index.php/CapOnt.

In addition, the physical storage of the capability type and non-rule-based capability parameter instances in a rational database is explained in Paper C.

C3: Policy-based reasoning

Policy-based reasoning is a support functionality related to the ability of adaptable service systems to take adaptation actions based on flexible and expressive behavioural specifications.

The policy-based reasoning applies a *generic reasoning procedure* to select appropriate adaptation actions among all existing adaptation actions, which are defined by the policies. This generic reasoning procedure solves a given query by transforming it through the repetitive application of equivalent *transformation rules* based on the Equivalent Transformation (ET) paradigm [ASM98]. The transformation rules can contain zero or more *variables*. During the transformation, the generic reasoning procedure also gives the transformation rule variables appropriate values, if possible. The generic reasoning procedure and its transformation, as illustrated in Figure 14, are thoroughly presented in Paper B. The query and transformation rules are represented by an XDD XML clause; thus, they are denoted as *query clauses* and *transformation rule clauses*, respectively. The transformation ends, and a desired answer (i.e., the appropriate adaptation actions) is obtained if the query clause is successfully transformed into at least one *final clause* that contains no variables.

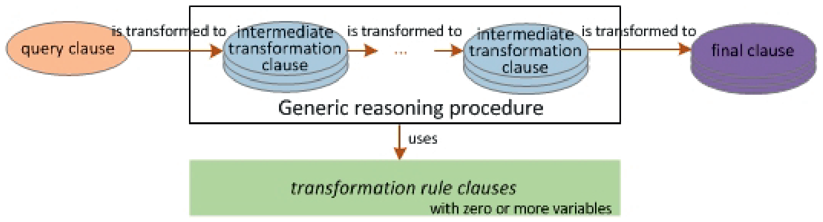

query clause
is transformed to
intermediate transformation clause
is transformed to
...
intermediate transformation clause
is transformed to
final clause
Generic reasoning procedure
uses
transformation rule clauses
with zero or more variables

Figure 14: Generic reasoning procedure.

This contribution is related to policy-based reasoning, and it consists of the *RM-based service component*, *generic policy system definitions*, and *dynamic policies*.

RM-based service component

The *RM-based service component* can provide the *policy-based reasoning*. The RM-based service component specification is a combination of an EFSM and a policy specification, as mentioned above in **C1**. The RM-based service component is then realised by an actor with RM executing these specifications. The RM-based service component functionality (RM_{SC}) can be formally defined as:

$$RM_{SC} \equiv E_{\Sigma} + R \qquad \textbf{(1)}$$

where E_{Σ} is the functionality of the actor executing the EFSM specification, and *R* is the functionality of the RM that is executing the policy specification. The actor providing the functionality E_{Σ} is also denoted as a *dedicated EFSM*. Its behaviour is generic and involves *activating the RM functionality when adaptation actions are needed*.

In this thesis, the functionality of the RM that executes the policy specification ***R*** can be defined by one of the following definitions: $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$. In the following subsections, R_1 and R_2 are first explained briefly. The difference between them is summarised. Then, the application domains of the RM-based service components based on both R_1 and R_2 are explained.

1) $\mathbf{R_1}$, as explained in Paper E, is defined as:

$$R_1 \equiv \{ \mathcal{Q}, \mathcal{F}, \mathcal{P}_1, \mathcal{T}, \xi_1, \Sigma_1 \} \qquad \textbf{(2)}$$

$$\mathcal{P}_1 \equiv \{ \mathcal{X}, \mathcal{A} \} \qquad \textbf{(3)}$$

where $\mathcal{Q}$ is a set of *queries*, $\mathcal{F}$ is *the generic reasoning procedure*, $\mathcal{P}_1$ is a *policy system*, $\mathcal{T}$ is a set of *system constraints*, ξ_1 is a set of *inherent system performances*, and Σ_1 is a set of *reasoning conditions*. The system constraints represent *variables* of the service system as well as the defined constraints and relationships between these variables. The reasoning conditions are classified into trigger conditions Σ_T and goal conditions Σ_G. The trigger conditions, Σ_T, define inherent states where adaptation actions are needed, and goal conditions Σ_G are used for the opposite situation. Since reasoning conditions are written as Boolean expressions, a goal condition's expression is simply the negation of a trigger condition's expression.

The policy system is a set of rules $\mathcal{X}$ and a set of actions $\mathcal{A}$. A rule $\mathcal{X}_i$ is composed of a Boolean expression based on the variables var in $\mathcal{T}$ and a set of associated actions $\mathcal{A}_i$ from the policy action set $\mathcal{A}$. A rule definition $\mathcal{X}_i$ which explicitly states $\mathcal{T}$ and $\mathcal{A}$ is:

$$\mathcal{X}_i \equiv (\text{Expression}(\text{var}), \mathcal{A}_i);\ \text{var} \in \mathcal{T}, \mathcal{A}_i \in \mathcal{A} \qquad \textbf{(4)}$$

In general, the variables in $\mathcal{T}$ can be variables of the required and inherent functionality and performance of services and capabilities, the required and inherent service income, or the available nodes, their capabilities, and system time. The expression (4) is then equivalent to expression (5), as follows:

$$\mathcal{X}_i \equiv (\text{Expression}(\overline{S_R}, \hat{S}_R, \overline{C_R}, \hat{C}_R, I_R, \overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I, \hat{C}_{A,n}; n=[1, N], \Gamma), \mathcal{A}_i);\ \mathcal{A}_i \in \mathcal{A} \qquad \textbf{(5)}$$

where $\overline{S_R}$ is the required service functionality set, $\hat{S}_R$ is the required service performance set, $\overline{C_R}$ is the required capability functionality set, $\hat{C}_R$ is the required capability performance set, I_R is the required service income, $\overline{S_I}$ is the inherent service functionality set, $\hat{S}_I$ is the inherent service performance set, $\overline{C_I}$ is the inherent capability functionality set, $\hat{C}_I$ is the inherent capability performance set, I_I is the inherent service income, $\hat{C}_{A,n}$ is a set of capabilities in the available nodes, and Γ is system time.

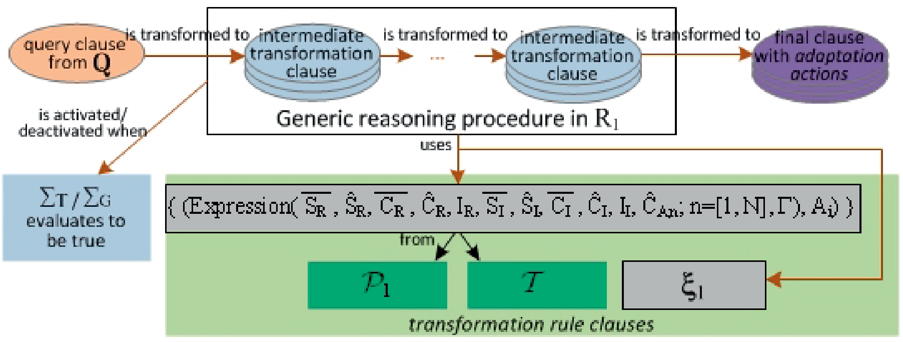

query clause from Q
is transformed to
intermediate transformation clause
is transformed to
...
intermediate transformation clause
is transformed to
final clause with adaptation actions
Generic reasoning procedure in R1
is activated/ deactivated when
ΣT / ΣG evaluates to be true
uses
{ (Expression(SR , ŜR, CR , ĈR, IR, SI , ŜI, CI , ĈI, II, ĈA,n; n=[1,N],Γ), Ai) }
from
P1
T
ξI
transformation rule clauses

Figure 15: Generic reasoning procedure in R₁.

Figure 15 illustrates the generic reasoning procedure as applied in R_1. The query clauses are $\mathcal{Q}$, and the transformation rule clauses are $\mathcal{P}_1$, $\mathcal{T}$, and ξ_1. Concerning $\mathcal{P}_1$ and $\mathcal{T}$, only $\{\mathcal{X}_i\}$, which includes the associated actions $\mathcal{A}_i$ and var, is used in the transformation (see expression (**5**) where $\mathcal{A}_i$ and var are present). The other actions in $\mathcal{A}$ and the other variables in $\mathcal{T}$ that are not a part of $\{\mathcal{X}_i\}$ are not used in the transformation. Thus, they do not affect the answer from the transformation, i.e., the appropriate adaptation actions.

2) R_2 is used in Papers F and G. R_2 is an extension of R_1, and is defined as:

$$R_2 \equiv \{ \mathcal{Q}, \mathcal{F}, \mathcal{P}_2, \xi_2 \} \quad \textbf{(6)}$$

$$\mathcal{P}_2 \equiv \{ p_i \} \quad \textbf{(7)}$$

$$p_i \equiv (\Sigma_i, X_i, \mathcal{A}_i) \quad \textbf{(8)}$$

where ξ_2 is ξ_1 plus inherent service income and $\mathcal{P}_2$ is a set of policies p_i. The policy specifications are based on the goal-based policy ontology in **C2**, where, according to this ontology, a policy p_i has (policy) conditions Σ_i, (policy) constraints X_i, and actions $\mathcal{A}_i$. Again, a constraint is a Boolean expression of required and inherent functionality and performance of services and capabilities, required and inherent service income, as well as available nodes, their capabilities, and system time. The expression (**8**) is then equivalent to expression (**9**), as follows:

$$p_i \equiv (\Sigma_i, \text{Expression}(\overline{S_R} , \hat{S}_R, \overline{C_R} , \hat{C}_R, I_R, \overline{S_I} , \hat{S}_I, \overline{C_I} , \hat{C}_I, I_I, \hat{C}_{A,n}; n=[1, N], \Gamma), \mathcal{A}_i) \quad \textbf{(9)}$$

Figure 16 illustrates the generic reasoning procedure applied in R_2. The query clauses are $\mathcal{Q}$, and the transformation rule clauses are $\mathcal{P}_2$ and ξ_2. Concerning $\mathcal{P}_2$, only the $\{(X_k, \mathcal{A}_k)\}$ from $\{ p_k \}$, where the Σ_k of p_k evaluates to be true, are used in the transformation. This means that the policies used each time the RM functionality is activated by the dedicated EFSM are only a subset of all the defined policies.

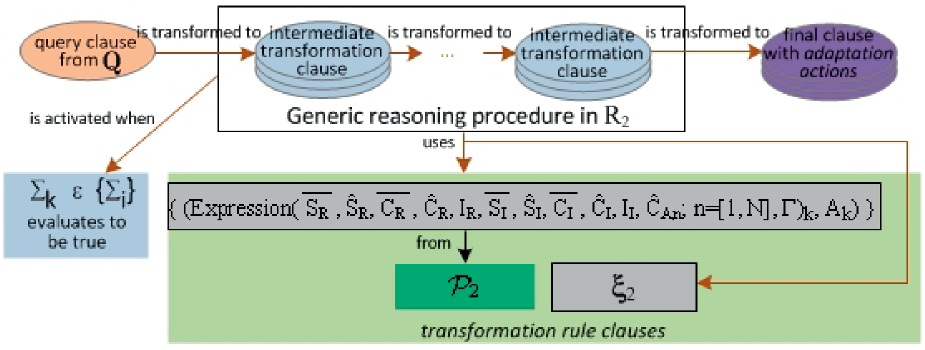

query clause from Q
is transformed to
intermediate transformation clause
is transformed to
...
intermediate transformation clause
is transformed to
final clause with *adaptation actions*
Generic reasoning procedure in R_2
is activated when
$\Sigma_k \varepsilon \{\Sigma_i\}$ evaluates to be true
uses
$\{(\text{Expression}(\overline{S_R}, \hat{S}_R, \overline{C_R}, \hat{C}_R, I_R, \overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I, \hat{C}_{An}; n=[1,N], \Gamma)_k, A_k)\}$
from
$\mathcal{P}_2$
ξ_2
transformation rule clauses

Figure 16: Generic reasoning procedure in R_2.

3) Difference between $\mathbf{R_1}$ and $\mathbf{R_2}$

The main difference between $\mathbf{R_1}$ and $\mathbf{R_2}$ is the *policies* on which they are based. The policies in $\mathbf{R_1}$ are based on CIM, and are represented by XML, RDF, and XML/XDD. The policies in $\mathbf{R_2}$ are defined by the goal-based policy ontology in **C2**, and are represented by OWL and OWL/XDD. Within these policies, the differences which affect the answer from the transformation, i.e., the appropriate adaptation actions, are the trigger conditions Σ_T and the (policy) conditions Σ_i. In $\mathbf{R_1}$, the trigger conditions Σ_T do not relate to any specific policies. If the Σ_T evaluates to be true, $\mathcal{F}$ will be activated and $\{\mathcal{X}_i\}$ from "all defined policies $\mathcal{P}_1$" will be used in the transformation. In $\mathbf{R_2}$, the conditions Σ_i relate to one or more specific policies. If a Σ_k evaluates to be true, $\mathcal{F}$ will be activated and "only the $\{(X_k, \mathcal{A}_k)\}$ from $\{p_k\}$ that holds this Σ_k" will be used in the transformation, where $\{p_k\} \in \mathcal{P}_2$. However, there is no goal condition concept in $\mathbf{R_2}$, so the dedicated EFSM must automatically deactivate $\mathcal{F}$ if the Σ_k evaluates to be false.

Concerning the difference between the Σ_1 and Σ_i as explained above, the definition R_1 can be represented by the definition R_2, but not vice versa. To represent R_1 by R_2, only one Σ_i is defined. The $\Sigma_i = \Sigma_T$, which relates to all defined policies in $\mathcal{P}_2$. That is, when Σ_i evaluates to be true, the $\{(X_i, \mathcal{A}_i)\}$ from "all defined policies in $\mathcal{P}_2$" are used in the transformation.

4) Application domains of the RM-based service components (based on both R_1 and R_2)

The use of the RM-based service components can be classified into two domains based on Σ_T and Σ_i, which again define the inherent states in which adaptation actions are needed. In the first application domain, Σ_T and Σ_i are not empty, while in the second application domain, Σ_T and Σ_i are empty.

For the first application domain, a *reasoning cluster* is defined as an independent unit with respect to policy-based reasoning. A reasoning cluster consists of a collection of EFSM-based service components with an associated reasoning system comprised of one or more RM-based service components. A dedicated EFSM from a RM-based service component regularly inspects the Σ_T or Σ_i. The E_Σ will invoke an EFSM-based service component in the reasoning cluster when adaptation actions are needed (i.e., when Σ_T evaluates to be true, or a Σ_k evaluates to be true). This EFSM-based service component will then call the RM in order to obtain the adaptation actions. In addition, the ξ_1 and ξ_2 are shared in the reasoning cluster. The variables in $\mathcal{T}$ can include the service component types in the reasoning cluster. For example, the adaptation actions are to change the present state of the EFSM, the local variables or behaviours of the EFSM-based service component itself (or other EFSMs), to manage the capabilities, and to change the dynamic policies used in other RM-based service components. Figure 17 illustrates these examples. The case that the adaptation actions are to change the dynamic policies will be explained in more detail in the sub-contribution dynamic policies section below.

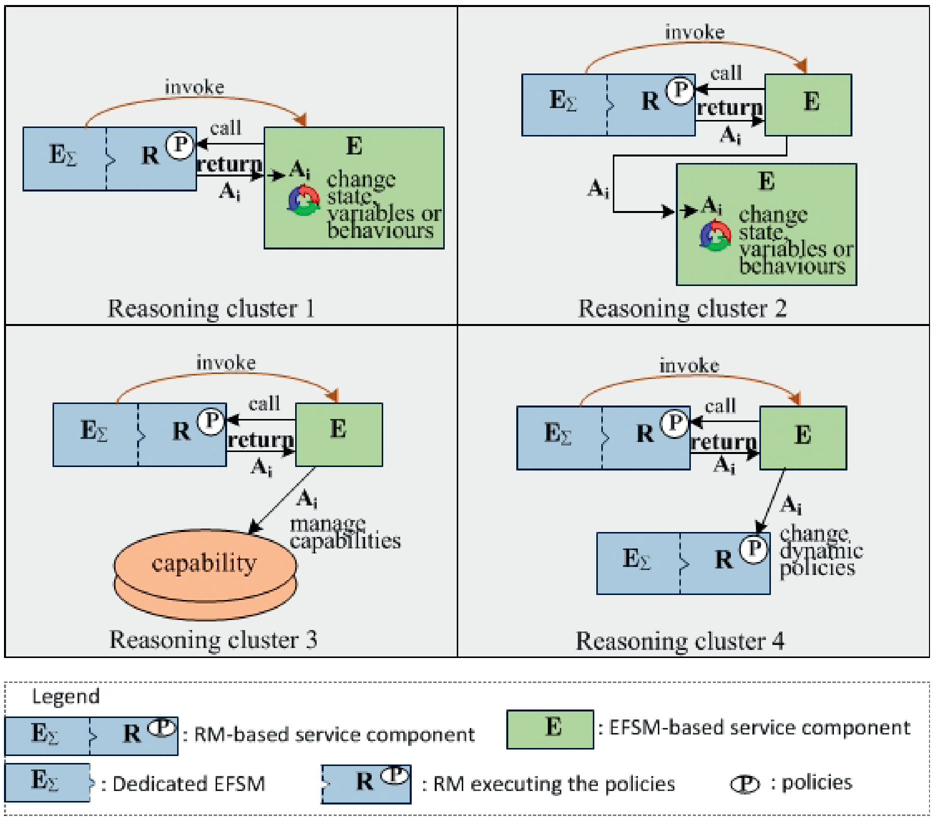


Figure 17: Examples of the use of RM-based service components in reasoning clusters.

For the second application domain, a dedicated EFSM is not needed because Σ_T and Σ_i are empty. An RM-based service component is used as a *traditional procedure call* for EFSM-based service components. In this application domain, the EFSM-based service components can call the RM by themselves when needed.

In the second application domain, where R_2 is used and the policies in $\mathcal{P}_2$ have no conditions, expression (**8**) is equivalent to expression (**10**) below:

$$p_i \equiv (X_i, \mathcal{A}_i) \qquad \textbf{(10)}$$

If the capabilities are focused and the actions $\mathcal{A}_i$, without the accumulated rewards and the operation costs, are to manage the capabilities, the policies will become capability-related simplified policies or service management functions in the capability ontology, as explained above. Thus, the capability ontology can enable the specifications for these policies. Because the service management functions are defined by rules, the policy-based reasoning functionality using the service management functions is denoted as *rule-based reasoning*.

Generic policy system definitions

Two policy systems ($\mathcal{P}_1$ and $\mathcal{P}_2$) and their formal definitions have been presented in this thesis, as discussed above. Both of these are generic, and are applicable to any adaptable service systems. An example where $\mathcal{P}_1$ is applied to a specific adaptable system that provides streaming services can be found in Paper E, and $\mathcal{P}_2$ is applied to the same system in Papers F and G. The policy specifications based on $\mathcal{P}_1$ and $\mathcal{P}_2$ are given in these corresponding papers.

Dynamic policies

Dynamic policies are changeable policies, as defined above in Section 1.1.4. Again, the dynamic policy changes can be on *entities of the policies*, on the *policy priorities*, and on the *policy usage statuses* (i.e., active, inactive, and deleted). The adaptation actions from the RM-based service components in this thesis can realise *all* of these changes. This means that these adaptation actions are to change the dynamic polices used in other RM-based service components. According to the learning mechanism explained in Section 1.1.4, the RM-based service components with these adaptation actions provide the same functionality as the learning mechanism.

The dynamic policy changes are based on *goodness criteria*, which are measures that indicate the ability of the policies or the adaptation actions defined by the policies that move the system towards a desired state. The *feedbacks* that are the inherent system performance as well as the inherent service income (the ξ_1 and ξ_2 in expressions (**2**) and (**6**)) are the basis from which the goodness criteria are calculated. The generic reasoning procedure has been extended to support the dynamic policies. That is, if the transformation gives several adaptation actions from different policies, only the actions with the highest priority, or the actions from the policies with active statuses are selected.

The changes on the policy usage statuses are presented in detail in Paper E, and the changes on the entities of the policies appear in Papers F and G. In Paper E, the RM-based service components that provide the same functionality as the learning mechanism are used, and the policy usage statuses are changed based on the *accumulated goodness scores*. In Papers F and G, AE-based service components with a learning mechanism are used. Here, the *accumulated rewards* of actions, which are the policy entities, are changed. In this case, the accumulated rewards are both goodness criteria and the priority of actions, because if there are several adaptation actions returned from the transformation, the actions with the highest accumulated reward will be selected. The action rewarding will be explained more fully in **C4**, below.

C4: Autonomic-based model

This contribution applies the principles and architecture of autonomic systems to adaptable service systems. A service component that is realised by an Autonomic

Element (AE) can be denoted as an AE-based service component. A service system can then be constituted by a collection of AE-based service components.

This contribution consists of *AE-based service component* and *action rewarding*, which is a learning mechanism used by AEs. AE-based service components can be found in Paper F, and action rewarding is highlighted in Papers F and G.

AE-based service component

The AE-based service component specification is a combination of two EFSM specifications, a policy, and a goal specification, as previously mentioned in **C1**. The AE-based service component consists of two actors. The first actor with RM and LM executes one of the two EFSM specifications, as well as the policy and goal specifications; the second actor, without RM and LM, executes the remaining EFSM specification. The AE-based service component functionality (AE_{SC}) can be formally defined as:

$$AE_{SC} \equiv E_M + R + L + E_C \qquad \textbf{(11)}$$

where E_M is the functionality of the first actor executing the EFSM specification, R is the functionality of the RM executing the policy specification, L is the functionality of the LM executing the goal specification, and E_C is the functionality of the second actor executing the remaining EFSM specification. In Paper F, the AE-based service component functionalities, as defined in expression (**11**), are classified into four functional modules: *Main Function*, *Strategist*, *Judge*, and *Communicator*. The Main Function, Strategist, Judge, and Communicator functionalities are equivalent to E_M, R, L, and E_C, respectively. These modules' behaviours and specifications can be found in detail in Paper F.

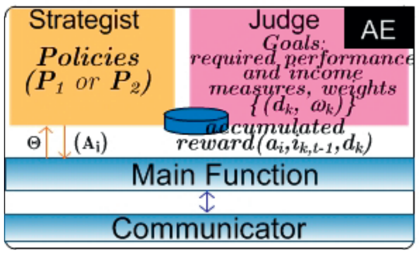


Figure 18: Autonomic Element model.

Figure 18 illustrates the AE model. A comparison of this AE model to other models will be presented below in Section 6. This AE model and the generic behaviours of each module, however, are designed to meet the individual and shared properties defined in Paper F. The individual properties are enabled on an AE, while the shared properties rely on the cooperation of several AEs.

The functionality of the RM executing the policy specification ***R*** in expression (**11**) can be realised by both R_1 and R_2. However, only the formal expression of R_2 using $\mathcal{P}_2$ is presented explicitly in Paper F. The experimental results in this paper, however, indicate that the AE-based service components *based on* R_2 can produce equal or better system performance and service income. This means that the adaptation actions applied by the AE-based service components based on R_2 are "more appropriate" than the adaptation actions applied by their counterpart (i.e., the AE-based service components based on R_1). The main reason why the adaptation actions given from R_1 and R_2 are different has been explained above in **C3**, subsection **3)** Difference between $\mathbf{R_1}$ and $\mathbf{R_2}$.

The autonomic service systems that consist of AE-based service components can be structured as *centralised* models, *hierarchical semi-centralised* models, or *decentralised* models. Some examples are depicted in Figure 19. However, only a centralised model is emphasised in Paper F.

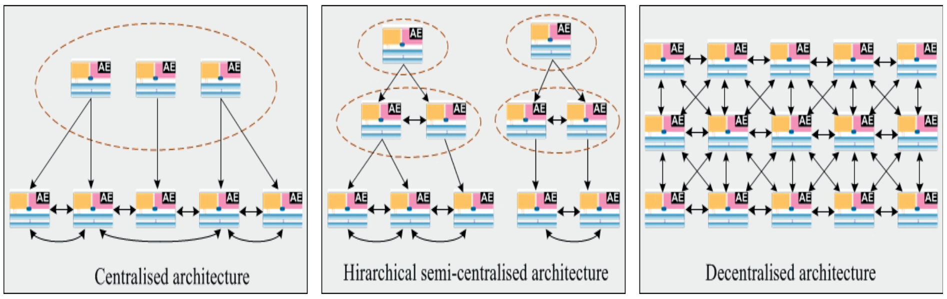

AE
Centralised architecture
Hirarchical semi-centralised architecture
Decentralised architecture

Figure 19: Structures of the autonomic service systems.

Action rewarding

Action rewarding is a *learning* mechanism that is provided by the Judge module to calculate rewards and accumulated rewards of actions. *An accumulated reward* of an action indicates the ability of the action to move the system from *an inherent state* towards *a desired state* defined by goal performance and income measures. An action can have several accumulated rewards, each of which is for a pair of states, one being inherent and the other desired. Accordingly, under an inherent and desired state, the generic reasoning procedure in the Strategist module selects the actions with the highest corresponding accumulated reward.

C5: Capability-related adaptation functionality realisation

This contribution deals with system architectures consisting of distributed EFSM-, RM-, and AE-based service components that realise the functionality components CSA, CSM, CC, CU, SM, PI, FD, PD, and PSC, as well as a part of AA and II in the service functionality architecture (see Figure 8). This contribution consists of three system architectures: a *two-level EFSM-based system architecture* for node and capability monitoring, a *RM-based system architecture* for capability parameter configuration, and an *AE-based service functionality system architecture*, as well as *agent's information model transformation and capability-based message specification*. Generic definitions and behaviours of the EFSM-, RM-, and AE-based service components in these system architectures are defined in **C1**, **C3**, and **C4**, respectively. Specific functionalities provided by these service components, as well as their inter-working and structural organisation are, however, required.

Concerning the realised functionality components in these architectures, humans must be involved in PI for the preliminarily installation of an EFSM-based service component in the nodes. This service component automatically downloads and installs platform software when these nodes are (re)started and when there is a new version of the platform software. The *agents* from the existing various network management frameworks are applied in these architectures to realise a part of the CSM and CC; thus, the information model transformation and capability-based message specification for the agents are included in this contribution.

Two-level EFSM-based monitoring system architecture

A system architecture consisting of EFSM-based service components and SNMP agents is presented in Paper D. This architecture focuses on *node and capability monitoring*, which is a part of the CSM; however, a part of the CSA, CC, CU, SM, PI, FD, PD, PSC, AA, and II are also realised with this architecture. The node and capability monitoring is decentralised and is realised by ordinary SNMP agents and the *two-level distributed EFSM-based service components*: the main monitoring managers and the intermediate monitoring manager. All messages, some necessary local variables of these managers, and their behavioural functions are defined. In Paper D, SNMP agents are used, but the system architecture can support the integration of agents from other network management frameworks.

This two-level EFSM-based service component structure provides *decentralisation* of the service component functionalities, and it can be applied to functions other than the CSM. Overall, this EFSM-based system architecture provides *adaptability* in the event of overload and fault situations, the *update of the platform software*, and *the EFSM specifications* for both the primary service functionalities and the management functionalities during runtime.

RM-based capability parameter configuration system architecture

A system architecture consisting of an RM-based service component, EFSM-based service components, and NETCONF agents is presented in Paper C. This architecture focuses on capability parameter configuration, which is a part of CC. However, this architecture also realises a part of the CSA and CSM. The *validation and settings of capability parameter values* are performed by *EFSM-* and *RM-based service components* that are used as a traditional procedure call for the EFSM. This RM-based service component is based on R_2 using the service management functions in the capability ontology. Here, the agents have the ability to (de)register nodes and capabilities. In Paper C, the NETCONF agents are used; however, this system architecture can support integration with agents from other network management frameworks.

This RM-based system architecture provides the *automatic discovery* of nodes and capabilities and the *automatic capability parameter configuration* based on the RM functionality. It also provides *flexibility* since the adaptation actions, the specifications for validation, and settings of capability parameter values can be added, modified, and removed during system runtime.

AE-based service functionality system architecture

A centralised system architecture consisting of AE-based service components is presented in Paper F. The agents from the existing various network management frameworks are also a part of this architecture. However, in Paper F, these agents are invisible. Here, the functionality components CSA, CSM, CC, CU, SM, PI, FD, PD, and PSC, as well as a part of AA and II are realised. The functionalities and inter-working of each AE-based service component in this system architecture are defined.

Because the AE is designed based on EFSM- and RM-based service components using EFSM and policy specification, this system architecture inherits the features of the EFSM-based and RM-based system architectures, as mentioned above in the previous sub-contribution sections.

Agent's information model transformation and message specification

The agents from the existing various network management frameworks are used in this thesis to provide the network management functionalities. Thus, the three system architectures mentioned above have the functionality to transform between the managed objects defined in these agents' information models and the capability ontology concepts (i.e., the capability types and the capability parameters). In paper C, the transformation between the NETCONF MIB in the YANG modelling language and the capability ontology concepts in OWL are presented.

In Paper A, the agents that apply the capability ontology as their information models are considered. The SOAP-based message specification to obtain, set, create, delete, subscribe, and unsubscribe the capabilities is proposed. This message specification is based on the WS-Management framework [DMT08].

C6: Service execution platforms, prototypes, and simulations

This contribution involves the validation and evaluation of the remaining contributions in **C1**–**C5**. This contribution consists of *service execution platforms*, *prototypes*, and *simulation software*. There are two prototypes implemented based on the service execution platforms, as follows: the *capability parameter configuration system* and the *adaptable monitoring system*. Two simulation software packages that model the behaviours of *adaptable file transfer systems* and *streaming service systems* are also presented. These are implemented with the Java programming language. The prototypes and simulation software execute in specifically designed scenarios, and provide some experimental results.

TAPAS core and management platforms

The service execution platforms have been explained above in Section 3.2. The implemented management platform is a new platform. The core platform has been revised and extended with new features as summarised in Section 3.3. The technical documents of these service execution platforms are available at http://tapas.item.ntnu.no/wiki/index.php/ TAPAS_Platform.

Capability parameter configuration system

A prototype of the RM-based capability parameter configuration system architecture in **C5** is implemented and presented in Paper C. The *capability ontology*, the *rule-based reasoning* functionality, and the *EFSM-* and *RM-based service components* are demonstrated. For the scenario in Paper C, SNMP IF-MIB [MK00] is used as the basis for the definitions of capability types and non-rule-based parameters. The automatic parameter value setting by the prototype can adapt the number of active web servers according to the number of user requests.

Adaptable file transfer system

An adaptable file transfer system is presented in Paper B. This system demonstrates *capability ontology*, *rule-based reasoning* functionality, and *EFSM-* and *RM-based service components*. For the scenario presented in Paper B, the capability types and parameters are based on SNMP Host Resource MIB [WG00] and IF-MIB [MK00]. This system dynamically (re)allocates the network bandwidth to the high priority services based on their inherent capability performance, i.e., the bandwidth utilisation.

Adaptable monitoring system

In Paper D, a prototype of the EFSM-based monitoring system architecture in **C5** is implemented and presented. The *EFSM-based service components* are demonstrated, wherein its specification supports movement in the event of overload and fault situations during runtime. For the scenario in Paper D where a service component failed, the functionality of this failed service component was first moved and reinstantiated in a new node, and then the co-operation with other service components was recovered.

Policy-based streaming service system

A system providing a streaming service is presented in Paper E to demonstrate the *EFSM-* and *RM-based service components* formed from R_1 and the policy model $\mathcal{P}_1$, as well as *policy-based reasoning* and *dynamic policies*. Dynamic policies and static policies (i.e., the non-changeable policies) are defined for the capability (re)allocation in the presented scenario. The dynamic policies could be suspended for a certain time period based on the accumulated goodness score. The inherent service performance and service income are compared in the situations when static policies, dynamic policies, and no policies are used for the capability reallocation.

Goal and policy-based streaming service system

The streaming service system used in Paper E was extended and presented in Papers F and G to demonstrate *goal-based policy ontology*, *AE-based service components*, and *action rewarding*. In order to compare the new proposed policy model $\mathcal{P}_2$ to the previous model $\mathcal{P}_1$, the same set of actions used in Paper E was applied in order to reallocate the capabilities under the same scenario. The inherent service performance and service income in both cases are compared. The study of the new proposed policy model in situations where there are different actions is also presented.

4.2 The realisation of the closed feedback loop systems

The service framework contributions can be used to implement the closed feedback loop systems Type A–E as follows:

- The pure algorithm system without policy and learning (Type A) consists of the EFSM-based service components using the EFSM specification in **C1**. An example of this is the adaptable monitoring system in **C6**. The adaptation actions are a part of the EFSM specifications. In the event of overload and fault situations that require adaptation actions, the possible adaptation action is fixed. This adaptation action is moving the service component functionality to a new random node that has the required capability functionality and performance. However, since updating the EFSM specifications during runtime is possible, the possible adaptation actions can be added and modified.

- The policy-based system without learning (Type B) consists of the EFSM- and RM-based service components based on R_1 in **C1** and **C3**. An example of this is the policy-based streaming service system that uses static policies in **C6**.

- The policy-based ontology system without learning (Type C) is made up of EFSM- and RM-based service components based on R_2 in **C1** and **C3**. The policy specification in R_2 is based on the goal-based policy ontology in **C2**. Moreover, the specifications of the capability-related simplified policies are based on the capability ontology in **C2**. An example of this is the capability parameter configuration system in **C6**.

- The policy-based system with learning (Type D) consists of AE-based service components derived from R_1 in **C4,** as well as the EFSM- and RM-based service components based on R_1 in **C1** and **C3** in cases where the adaptation actions change the dynamic polices. An example of this is the policy-based streaming service system that uses dynamic policies in **C6**.

- The policy-based ontology system with learning (Type E) incorporates AE-based service components based on R_2 in **C4,** as well as the EFSM- and RM-based service components built from R_2 in **C1** and **C3** in cases where the adaptation actions change the dynamic polices. The goal-based policy ontology and the capability ontology in **C2** are required for this system type. An example of this is the goal-based policy streaming service system in **C6**.

Figure 20 illustrates the relationship between the contributions and the closed feedback loop systems. This figure also visualises the *flexibility* concerning the updates during system runtime and the *effectiveness* with respect to the reasoning of each type. Types D and E provide more flexibility than the other types because only incremental changes in terms of the policy and/or goal specification are possible. The policy-based reasoning of the systems that use the ontologies is more powerful than their counterparts without the ontologies. The reason for this is explained in **C2** of Section 3 above.

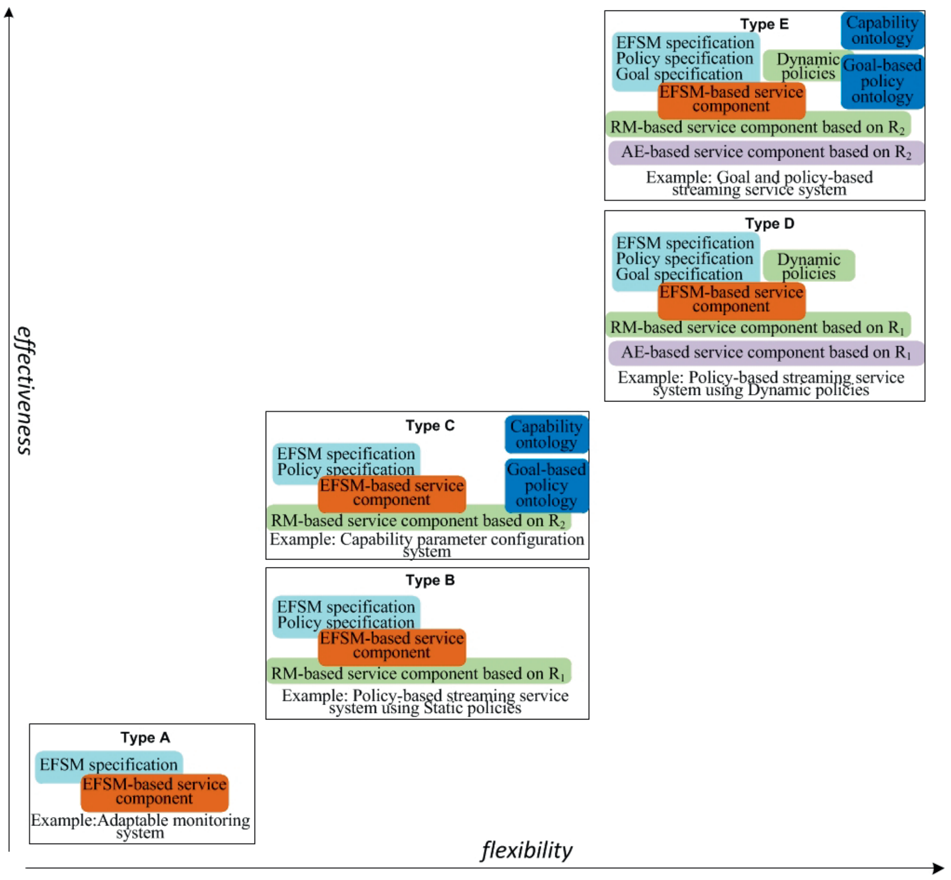

effectiveness
flexibility
Type A
EFSM specification
EFSM-based service component
Example:Adaptable monitoring system
Type B
EFSM specification
Policy specification
EFSM-based service component
RM-based service component based on R1
Example: Policy-based streaming service system using Static policies
Type C
Capability ontology
EFSM specification
Policy specification
EFSM-based service component
Goal-based policy ontology
RM-based service component based on R2
Example: Capability parameter configuration system
Type D
EFSM specification
Policy specification
Goal specification
Dynamic policies
EFSM-based service component
RM-based service component based on R1
AE-based service component based on R1
Example: Policy-based streaming service system using Dynamic policies
Type E
Capability ontology
EFSM specification
Policy specification
Goal specification
Dynamic policies
Goal-based policy ontology
EFSM-based service component
RM-based service component based on R2
AE-based service component based on R2
Example: Goal and policy-based streaming service system

Figure 20: Relationship between the close feedback loop systems and contributions.

4.3 The realisation of the problem statements

This section summarises the relationship between the problem statements **P1–P5** and the service framework contributions. The relationship between the contributions and the papers in Part II where they are presented is also summarised. Figure 21 below illustrates these relationships.

Concerning **P1**, the *ontology-based computing architecture* with new and revised concepts in **C1** is applied for the specification and execution of EFSM-, RM-, and AE-based service components. These concepts are capability types, capability parameters, EFSM specification, policy specification, goal specification, RM, LM, and role figure generic states. The capability concepts appear in all papers. The EFSM specification is presented in Papers D and F, the policy specification and RM in Papers E, F, and G, and the goal specification, LM, and role figure generic states in Papers F and G.

Concerning **P2**, the *capability* and *goal-based policy ontologies* in **C2** are applied. The capability ontology is the foundation for the capability parameter configuration specification, the capability functionality, and performance requirements. The goal-based policy ontology is the basis for the policy and goal specifications. The capability ontology is presented in Papers A, B, and C, while the goal-based policy ontology appears in Papers F and G. These ontologies are the *unified models* for both non-rule-based and rule-based concepts. They also support *interoperability*, *aggregation*, and *powerful reasoning*. OWL and OWL/XDD are used to represent these ontology concepts. The examples for these ontologies are available at http://tapas.item.ntnu.no/wiki/index.php/CapOnt.

In terms of **P3**, the EFSM- and RM-based service components can implement *all* the functions corresponding to capability-related adaptation. The generic definitions and movement features of the EFSM-based service components are defined in **C1**. The generic definitions and policy-based reasoning functionality of the RM-based service components, the generic policy system definitions, as well as the dynamic policies (in which the changes are made by the RM-based service components) are defined in **C3**. The RM-based service components appear in Papers B, C, and E, while the generic policy system definitions and the dynamic policies are presented in Paper E, F, and G.

The EFSM- and RM-based system architectures have been defined in **C5** to satisfy the specific functionalities of the needed service components for the realisation of the required functionalities for capability-related adaptation. These system architectures are presented in Papers D and C, respectively. In these architectures, the agents from the various existing network management frameworks are applied for the network management functionalities. The transformation between the managed objects in the agents' information models and the capability concepts is presented in Paper C. In the event that the capability ontology is the agent's information model, the message specification to obtain, set, create, delete, subscribe, and unsubscribe the capabilities is used, as presented in Paper A.

The use of the EFSM- and RM-based service components with their mechanisms and features provides *adaptability* in the event of overload and fault situations, *flexibility* to update service functionalities during system runtime, and *reasoning ability*, which results in, for example, the automatic modification of the dynamic policies, EFSM states, local variables, and behaviours.

For **P4**, the Autonomic Element model, wherein its functional modules are defined by EFSM, policy, and goal specifications, is defined in **C4**. The learning mechanisms of AE are derived from the Strategist module when the adaptation actions are to change the dynamic polices used in other AEs, as well as the Judge module (i.e., action rewarding as defined in **C4**). The AE-based service functionality system architecture defined in **C5** demonstrates the realisation of the capability-related adaptation functionalities. The AE model and the AE-based system architecture are presented in Paper F, while action rewarding is presented in Papers F and G.

This Autonomic Element model is designed to achieve the *individual* and *shared properties* as defined in Paper F. Moreover, because the EFSM- and RM-based service components using EFSM- and policy specifications are the foundation for AE, the use of AE-based service components provides the same advantages as the EFSM- and RM-based service components, as mentioned above.

Concerning **P5**, the TAPAS core platform and management platform have been implemented and are presented in **C6**. The technical documents of these platforms are available at http://tapas.item.ntnu.no/ wiki/index.php/ TAPAS_Platform. In addition, two prototypes and two simulation software packages used to validate and evaluate the contributions in **C1**–**C5** are presented. Prototypes that are implemented based on the TAPAS core platform and management platform can be found in Papers C and D, while the simulation software appear in Papers B, E, F, and G. Both the prototypes and the simulation software are executed in specifically designed scenarios and provide some experimental results from which we can conclude that the contributions in **C1**–**C5** meet the problem statements.

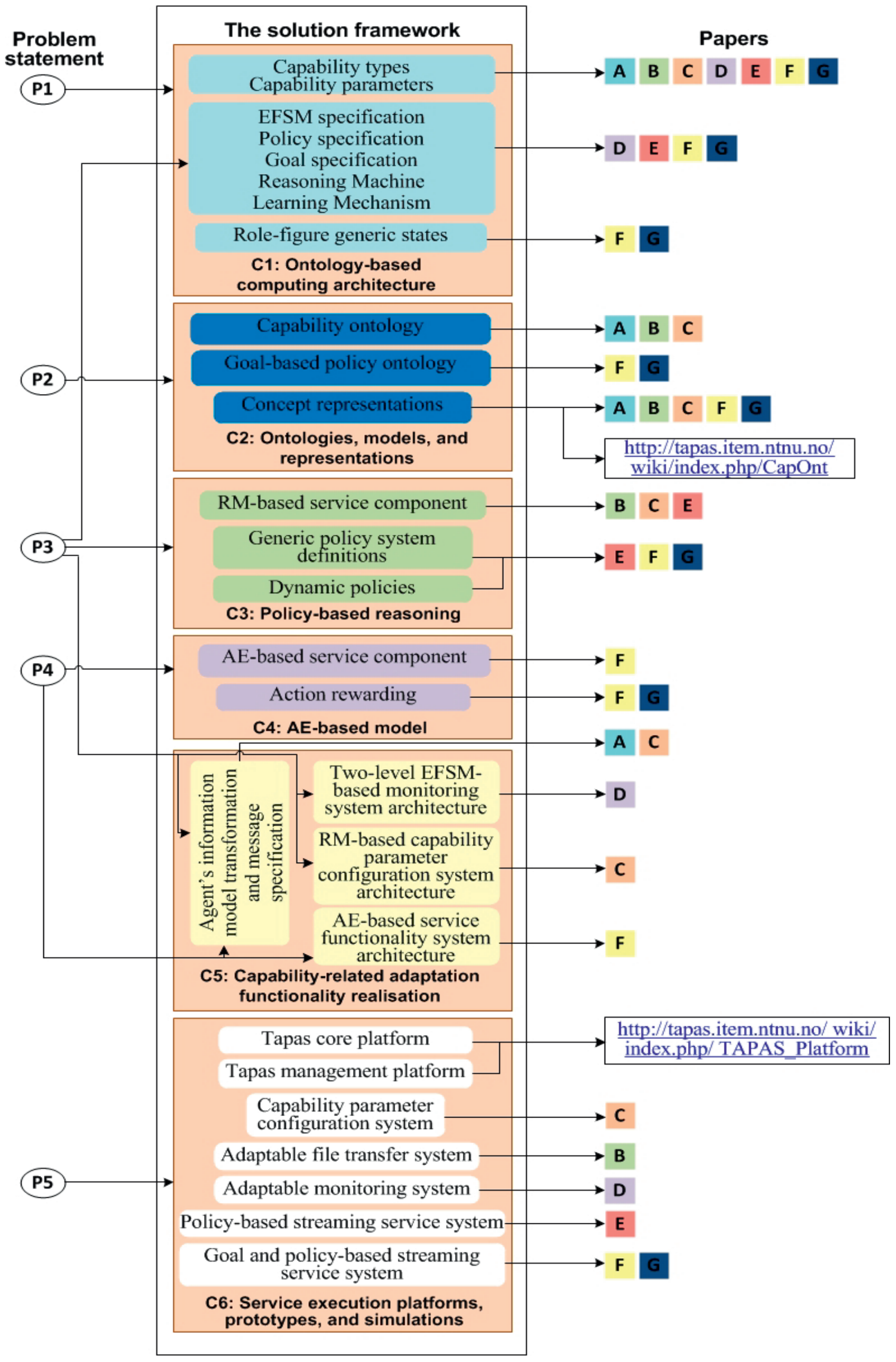

Problem statement
P1
P2
P3
P4
P5
The solution framework
Capability types
Capability parameters
EFSM specification
Policy specification
Goal specification
Reasoning Machine
Learning Mechanism
Role-figure generic states
C1: Ontology-based computing architecture
Capability ontology
Goal-based policy ontology
Concept representations
C2: Ontologies, models, and representations
RM-based service component
Generic policy system definitions
Dynamic policies
C3: Policy-based reasoning
AE-based service component
Action rewarding
C4: AE-based model
Agent's information model transformation and message specification
Two-level EFSM-based monitoring system architecture
RM-based capability parameter configuration system architecture
AE-based service functionality system architecture
C5: Capability-related adaptation functionality realisation
Tapas core platform
Tapas management platform
Capability parameter configuration system
Adaptable file transfer system
Adaptable monitoring system
Policy-based streaming service system
Goal and policy-based streaming service system
C6: Service execution platforms, prototypes, and simulations
Papers
A B C D E F G
D E F G
F G
A B C
F G
A B C F G
http://tapas.item.ntnu.no/wiki/index.php/CapOnt
B C E
E F G
F
F G
A C
D
C
F
http://tapas.item.ntnu.no/ wiki/ index.php/ TAPAS_Platform
C
B
D
E
F G

Figure 21: Relationship between the problem statements, contributions and papers in Part II.

5 Description of the papers in Part II

This section summarises the seven papers included in Part II of this thesis. The contributions of each paper are also presented here.

5.1 Paper A – Ontology-based Capability Self-Configuration

P. Thongtra, F.A. Aagesen, and N. Sriharee

in *Proc. 14th Eunice Open European Summer School on which networks for which services*, Brest, France, September 2008.

Abstract

Capability configuration management is the allocation, re-allocation and de-allocation of capabilities. A capability is here defined as an inherent physical property of a network node which is a basis for implementing networked services. Capability configuration management requires a specification of the capability configuration management process. This paper proposes a framework for capability configuration process specification (CCPS) production and execution. CCPS is a set of actions with related parameters and is used to configure the capability objects of the network nodes prior to service deployment and instantiation. A CCPS is based on Web services.

The production of CCPS is based on capability requirements defined for the networked services to be deployed and instantiated. The framework has an ontological reasoning ability. A case study to configure a VLAN connection using switches is also provided.

Contributions

- **C1**{Capability types}
- **C2**{Capability ontology; Concept representations}
- **C5**{Agent's message specification}

Summary

This paper presents a framework for the automatic production and execution of the CCPS according to the requirements of the capabilities. The CCPS consists of SOAP-based messages, each of which contains actions and related parameters for the obtaining, setting, subscribing, unsubscribing, logical creation, and deletion of the capabilities.

The framework consists of an information model, a functional organisation model, and a communication model as follows:

- *Information model:* The capability ontology is applied as the information model, and OWL is used to represent the ontology concepts. However, the capability ontology presented in this paper is in its initial version. The capability types and some of their attributes were defined in this initial version. However, the capability parameter concept was not yet defined explicitly.

- *Functional organisation model:* This model consists of three EFSM-based service components, the agents executing in the nodes, the capability ontology repository, and the web service registry. The agents in this paper apply the capability ontology and register the services for managing the capabilities to the web service registry. An EFSM-based service component denoted as Query Manager transforms the capability requirements into SPARQL queries, and then with the queries, validates which nodes whose capabilities already meet, or are able to meet, the requirements.

- *Communication model:* This model is related to the SOAP-based message specification that is based on the WS-Management framework.

The main focus of this paper is the capability parameter configuration, which is the same as Paper C, even though the capability parameters were not yet explicitly defined. The actions on the capability types and parameters include the obtaining, setting, subscribing, unsubscribing, logical creation, and deletion of the capabilities. In addition to the capability parameter configuration, which is a part of the CC, sections of the CSA and CSM are considered.

A case study is presented. It demonstrates the automatic production of the CCPS for a set of switches in a VLAN. The capability types used as examples are memory, operating system, and VTP protocol. From this, a CCPS consisting of two SOAP messages is generated. Each of these messages sets the VTP protocol mode in a selected switch, as required.

5.2 Paper B – Capability Ontology in Adaptable Service System Framework

P. Thongtra and F.A. Aagesen

in *Proc. IARIA/IEEE 5th Int. Multi-Conf. Computing in the Global Information Technology (ICCGI 2010)*, Valencia, Spain, September 2010.

Abstract

This paper presents a Capability Ontology (CapOnt) and a rule-based reasoning mechanism, which support service management within adaptable service systems. The ontology concepts comprise capability types, capability parameters and service management functions related to capabilities. Capability parameter values can be defined by constraints on other capability parameters. The service management functions included in the ontology are Capability Administration, Capability Configuration, Capability Allocation and Capability Performance Diagnosis. The service management functions are defined by rules consisting of constraints and management actions. The ontology concepts are represented in OWL (Web Ontology Language) and OWL/XDD (XML Declarative Description Language) – a rule-oriented knowledge representation. An intelligent conference room example with simulation results, that demonstrates the CapOnt and the rule-based reasoning, is also presented.

Contributions

- **C1**{Capability types and capability parameters}
- **C2**{Capability ontology; Concept representations}
- **C3**{RM-based service component}
- **C6**{Adaptable file transfer system}

Summary

In this paper, the capability ontology and the rule-based reasoning provided by the RM-based service components are presented. The generic reasoning procedure and its transformation are also explained in detail.

Here, the capability ontology has been revised and extended from the initial version in Paper A. Specifically, capability parameters and service management functions have been added. The service management functions are capability-related actions with constraints, and accordingly, the actions presented in this paper realise parts of CSA, CC, CU, and PD. In addition to OWL representing the ontology non-rule-based concepts, OWL/XDD is introduced to represent the ontology rule-based concepts.

For the validation and evaluation of the capability ontology and rule-based reasoning, an adaptable file transfer system in a conference room is presented. This system consists of five EFSM-based service components and a RM-based service component, which is used as a traditional procedure call for one of the EFSM-based service components. In the presented scenario, the file transfer services are classified into three different classes based on the type of transferred file. When the number of users increases and the network bandwidth utilisation becomes too high, only the high priority services will be available and will operate regularly.

The capability types and parameters considered in the scenario are based on SNMP Host Resource MIB [WG00] and IF-MIB [MK00]. Two rule-based capability parameters, which are the performance parameter Utilization and the inference parameter Compatibility, are defined. Five service management functions for CSA, CC, and CU are also defined. The system provides the *adaptation* in case the bandwidth utilisation is greater or lower than the limits. In such a case, the EFSM-based service component receives the suggested adaptation actions from the RM-based service component to disable or enable low priority services. The bandwidth is then (re)allocated to the high priority services. The experimental results show that when the service management functions for the CU are applied, the system can more efficiently (re)allocate the bandwidth to the higher priority services.

5.3 Paper C – OWL-based Node Capability Parameter Configuration

P. Thongtra, F.A. Aagesen, and K. Dittawit

in *Proc. 18th EUNICE/IFIP WG 6.2, 6.6 Int. Conf. Information and Communications Technologies*, Budapest, Hungary, August 2012. *Lecture Notes in Computer Science (LNCS) 7479*, pp. 124-135, 2012.

Abstract

Node capability parameter configuration is the validation and settings of node capability parameter values according to node capability parameter configuration specification (CapSpc). A node capability is a property of a node required as basis for service implementation. This paper presents a Node Capability Parameter Configuration System (CapCon). Node Capability Ontology (CapOnt) is the basis for CapCon. This paper has focus on CapCon in network management. CapSpc specifies required types, parameters, and parameter values for the node capabilities. OWL and OWL/XDD are used to represent the ontology concepts. The NETCONF framework is applied for the network management functionality. A prototype implementation and a case study including experimental results are presented.

Contributions

- **C1**{Capability types and capability parameters}
- **C2**{Capability ontology; Concept representations}
- **C3**{RM-based service component}
- **C5**{RM-based capability parameter configuration system architecture; Agent's information model transformation}
- **C6**{Capability parameter configuration system}

Summary

This paper presents CapCon, which enables automatic node capability parameter configuration. Two aspects, including system architecture and node capability ontology, are discussed as follows. It should be noted that in this paper, the capability ontology, capability types, capability parameters, and capability parameter configurations are prefixed by "node" to prevent confusion with the NETCONF capability from the NETCONF network management framework [Enn06].

- *System architecture:* The system architecture consisting of a RM-based service component, EFSM-based service components, two repositories, and NETCONF agents is presented. The specific functionalities and interworkings provided by these service components and agents are explained. The validation and settings of the capability parameter values are performed by an EFSM- and RM-based service component that is used as a traditional procedure call for the EFSM. The agents have functionalities to (de)register nodes and capabilities.

- *Node capability ontology:* In addition to the definitions and the representations of node capability ontology concepts (i.e., the same as Paper B), the projection of the node capability types and non-rule-based parameters from SNMP MIBs as well as the transformation between these concepts and NETCONF MIBs are presented. The physical storage of the instances of the ontology concepts in a rational database is also explained. This paper shows that the node capability ontology is the foundation for the node capability parameter configuration specification (CapSpc), which specifies the required types, parameters, and parameter values for the node capabilities.

The main focus of this paper is the same as Paper A, i.e., the capability parameter configuration. A part of CSA and CSM are also realised by the presented system architecture in this paper.

The prototype CapCon has been implemented based on service execution platforms. In the case study, CapCon integrates with a real web-based application. CapCon dynamically connects or disconnects web servers based on the number of user requests. The agents are based on the YUMA toolkit [YUM11]. The YUMA agent functionalities have been extended to support NETCONF IF-MIB and to provide the registration and deregistration of nodes. SNMP IF-MIB [MK00] is used as the basis for the definitions of capability types and non-rule-based parameters. A rule-based parameter, ifInUtilisation, the same as the parameter Utilisation in Paper B, is applied. Two service management functions that reconnect or disconnect the nodes are also defined. During system runtime, the automatic parameter value setting by CapCon adapts the number of active web servers.

Experiments have also been conducted to further evaluate the automatic parameter value setting. The evaluation is based on the level of energy conservation achieved from the disconnection of web servers. Some significant results from the experiments are presented.

5.4 Paper D – An Adaptable Capability Monitoring System

P. Thongtra and F.A. Aagesen

in *Proc. IARIA/IEEE 6th Int. Conf. Networking and Services (ICNS 2010)*, Cancun, Mexico, March 2010.

Abstract

SNMP-based monitoring systems are founded on a platform with precompiled agents in the managed node. Software updates and re-configuration are not flexibly handled. This paper presents a solution for monitoring by use of a service framework for adaptable service systems in cooperation with traditional SNMP agents. A two-level manager system consisting of Main monitoring managers and Intermediate monitoring managers executes on TAPAS (Telematics Architecture for Play-based Adaptable Systems) platform for adaptable service systems. The Intermediate monitoring managers are federated managers that communicate with ordinary precompiled SNMP agents by using SNMP protocol. The manager part of the monitoring system will have flexibility both with respect to software updates and re-configuration. The managers are TAPAS role figures. Role figures can be deployed, instantiated and moved during run-time. This feature can be used in overload and fault situations. Some experimental results are presented for cases where the Main monitoring manager is autonomously moved after failure.

Contributions

- **C1**{Capability types; EFSM specification}
- **C5**{Two-level EFSM-based monitoring system architecture}
- **C6**{Adaptable monitoring system}

Summary

In this paper, a system architecture for node and capability monitoring based on TAPAS architectures and service execution platforms, as well as ordinary SNMP agents, is presented. This system architecture consists of EFSM-based service components and precompiled SNMP agents. Concerning the EFSM specification in the TAPAS computing architecture, the concept of a stable state is mentioned explicitly to support the service component movement. A stable state is one in which a service component can move safely; this means that its behaviours are suspended for a short period, and are reinstantiated in a new node with the present state, local variables, and waiting received messages.

This system architecture focuses on node and capability monitoring, which is a part of CSM, as well as a part of CSA, CC, SM, FD, AA, and II. Here, the node and capability monitoring is decentralised, and is realised by the SNMP agents and two-level distributed EFSM-based service components denoted as the main monitoring managers and the intermediate monitoring managers. The intermediate monitoring managers are federated managers that communicate with the SNMP agents by using SNMP protocol. All messages, some necessary local variables, and their behavioural functions are defined. In terms of the other functionality components, FD is, for example, distributed between the main monitoring managers, the intermediate monitoring managers, and the fault diagnosis manager. HeartBeat messages are used to indicate that a cooperating service component is alive. This EFSM-based system architecture provides *adaptability* in the event of overload and fault situations, as well as the *update of platform software* and *EFSM specification* during runtime.

The monitoring system prototype has been implemented based on the service execution platforms. The experiment carried out comprises where the main monitoring manager failed. The functionality of the failed manager was moved and reinstantiated in a new node, and then the co-operation with the intermediate monitoring manager was recovered. The experimental results give the average time to detect the failure and the average time to reinstantiate the failed manager and to resume its operation.

5.5 Paper E – Towards Policy-Supported Adaptable Service Systems

P. Supadulchai, F.A. Aagesen, and P. Thongtra

in *Proc. 13th Eunice Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services*, Twente, Netherlands, June 2007. *Lecture Notes in Computer Science (LNCS) 4606*, pp. 128-140, 2007.

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.

Contributions

- **C1**{Capability types; Policy specification and Reasoning Machine}
- **C3**{RM-based service component; Generic policy system definitions; Dynamic policies}
- **C6**{Policy-based streaming service system}

Summary

This paper presents a policy-supported architecture for adaptable service systems. The two aspects covered are:

- *Service system framework*
- *Adaptation mechanisms*

The service system framework aspect is comprised of necessary concepts and a general component structure. According to the general component structure, a service system consists of executing service components, which can be either the EFSM-based service component or the RM-based service component. In this paper, the RM-based service component is based on R_1 (see expression (**2**)).

The adaptation mechanism aspect concerns the use of appropriate policies to control service systems when adaptation actions are needed. The adaptation mechanisms can be based on static or dynamic policies. The static policies are unchangeable, while the dynamic policies can be changed. In this paper, the dynamic policies' usage statuses are changed based on their accumulated goodness scores, and accordingly, the dynamic policies can be suspended for a certain time period.

Generally, the application domains of the RM-based service components can be classified into two domains, as follows:

i) used in a reasoning cluster that consists of a collection of EFSM-based service components with an associated reasoning system that is composed of one or more RM-based service components, or

ii) used as a traditional procedure call for EFSM-based service components.

For the context of this paper, which focuses on *capability allocation adaptation*, the adaptation actions from the RM-based service components in application domain i) above are specialised for: a) Node selection prior to the service component deployment and instantiation, b) Capability usage (re)allocation, and c) the changes in the dynamic policies.

In this paper, a policy-based streaming service system is also presented to illustrate the use of the proposed architecture and the potential advantage of using dynamic policies. In the designed scenario, the users of the streaming system belong to two classes, where the QoS of each class is described by SLA. The users of the class with a higher priority have more allocating capabilities and privileges in using the service. A set of actions to reallocate the capabilities is then defined. This scenario includes capability reallocation using static policies, dynamic policies, and no policies. The experimental results show that the service system operated under dynamic policies gives superior or equal service income to the other cases under any traffic load. However, the service system needs a certain period of time (denoted as learning time) to learn the consequences of the dynamic policies in order to provide these superior results.

5.6 Paper F – An Autonomic Framework for Service Configuration

P. Thongtra and F.A. Aagesen

in *Proc. IARIA/IEEE 6th Int. Multi-Conf. Computing in the Global Information Technology (ICCGI 2011)*, Luxembourg, June 2011.

Abstract

An autonomic framework for service configuration functionality is proposed. The framework has goals and policies. Goals express required performance and income measures. Policies define actions in states with unwanted performance and income measures. All functionality is executed by autonomic elements (AEs) that have ability to download and execute behavior specifications during run-time. An AE has several generic functionality components. Two important generic components of an AE are Judge and Strategist. A Strategist selects actions in a state with unwanted performance and income measures to reach a state defined by goal performance and income measures. A Judge gives rewards to actions based on the ability to move towards a state with goal performance and income measures. The Strategist's selection of actions is based on the rewards given by the Judge. AE functionality is realized by the combination of Extended Finite State Machines (EFSM), a Reasoning Machine (RM) and a Learning Machine (LM). A case study of an adaptable streaming system is presented. Using the proposed model, the streaming system can select actions for capability allocation adaptation more appropriately as evaluated by the performance and income measure results.

Contributions

- **C1**{Capability types and capability parameters; EFSM, policy and goal specification, Reasoning Machine, and Learning Mechanism; Role figure generic states}
- **C2**{Goal-based policy ontology; Concept representations}
- **C3**{Generic policy system definitions; Dynamic policies}
- **C4**{AE-based service component; Action rewarding}
- **C5**{AE-based service functionality system architecture}
- **C6**{Goal and policy-based streaming service system}

Summary

This paper presents an autonomic framework for adaptable service systems. In this framework, a service component is realised by an Autonomic Element (AE). A service system is then composed of a collection of AEs. The AE behaviours can be defined by two EFSM specifications, a policy, and a goal specification.

The three aspects covered in this paper are:

- *Goal-based policy ontology*,
- *Autonomic Element model*, and
- *AE-based service functionality architecture.*

Goal-based policy ontology defines the common concepts of goals, policies, and inherent states. This ontology is the foundation for policy specification and goal specification.

The AE model is designed based on the AE individual and shared properties. The individual properties are obtained by an AE, while the shared properties are achieved through the co-operation of AEs. The AE functionalities are classified into four functional modules: Main Function, Strategist, Judge, and Communicator. The Main Function coordinates the functionality of an AE. The Strategist selects the appropriate adaptation actions to be used by the Main Function in a state with unwanted performance and income measures. The Judge gives rewards to the selected adaptation actions, and the Communicator handles message sending and receiving on behalf of the Main Function. Action rewarding is a learning mechanism that calculates the rewards and the accumulated rewards of selected adaptation actions. An accumulated reward indicates the ability of the action to move the service system from an inherent state towards a goal state.

The functionality of the Main Function is realised by an actor executing an EFSM specification, the Strategist functionality by a RM executing a policy specification, the Judge functionality by a LM executing a goal specification, and the Communicator functionality by an actor executing an EFSM specification. The Strategist functionality opens for both R_1 and R_2 (see expressions (**2**) and (**6**)), even though only the formal expression of R_2 is presented in this paper.

The service functionality architecture consists of specific AEs handling management functionalities as well as primary service functionalities. The functionalities and interworkings of each AE are defined here.

In this paper, the streaming service system used in Paper E is extended. The intention is to compare the new proposed policy model with the previous model in Paper E, and to study the new proposed policy model in situations when there are different actions. For the purpose of comparison, the same set of actions as in Paper E is applied to reallocate the capabilities. The experimental results show that the proposed policy model produces better results, especially during high traffic load. For the study of the proposed policy model, an extra action is defined. The experimental results also show that when there are more actions, the service system selects the most appropriate action and has the potential to reach the defined goals faster.

5.7 Paper G – On Capability-related Adaptation in Networked Service Systems

F.A. Aagesen and P. Thongtra

Int. Journal of Computer Networks & Communications (IJCNC), vol. 4, no. 4, July 2012.

Abstract

Adaptability is a property related to engineering as well as to the execution of networked service systems. This publication considers issues of adaptability both within a general and a scoped view. The general view considers issues of adaptation at two levels: 1) System of entities, functions and adaptability types, and 2) Architectures supporting adaptability. Adaptability types defined are capability-related, functionality-related and context-related adaptation. The scoped view of the publication is focusing on capability-related adaptation. A dynamic goal-based policy ontology is presented. The adaptation functionality is realized by the combination of extended finite state machines, reasoning machines and learning mechanisms. An example case demonstrating the use of a dynamic goal-based policy is presented.

Contributions

- **C1**{Capability types and capability parameters; EFSM, policy and goal specification, Reasoning Machine, and Learning Mechanism; Role figure generic states}
- **C2**{Goal-base policy ontology; Concept representations}
- **C3**{Generic policy system definitions; Dynamic policies}
- **C4**{Action rewarding}
- **C6**{Goal and policy-based streaming service system}

Summary

In this paper, the issues of adaptability of networked service systems both within a general view and a scoped view are presented. The general view consideres the issues of adaptation at two levels: 1) System of entities and functionalities related to the service system life cycle, and adaptability types with required adaptability functionality, and 2) Architectures supporting adaptability. Concerning the first level, the functionalities related to the service system life cycle are analysed. The adaptability types defined are capability-related, functionality-related, and context-related adaptation. The functionalities required for each of these adaptability types are summarised.

Concerning the second level, the computing architecture and the TAPAS service functionality architecture can support all of the defined adaptability types. The computing architecture opens for service systems defined by flexible combination of EFSM, goal, and policy specifications combined with learning mechanisms. However, the computing architecture presented in this paper is a generalised TAPAS computing architecture (i.e., the play view has been removed).

The scope view considers capability-related adaptation. The goal-based policy ontology and the case study from Paper F are presented. The static and dynamic models of the goal-based policy ontology are mentioned explicitly in this paper. The static model is the original goal-based policy ontology model found in Paper F, which does not include the concept operation cost and accumulated rewards. The static model can be made dynamic by the Learning Mechanism (LM), which calculates both the rewards and the accumulated rewards. The operation cost, however, is open to updates; its value can be fixed or changed by the LM as well. The case study from Paper F, which is also presented in this paper, is only for the study of the policy model based on the goal-based policy ontology.

6 Related works

This section briefly presents paradigms and technologies that are relevant to the service framework in this thesis. These include autonomic systems and communications, as well as policy-based network management. Moreover, some research papers related to the presented paradigms are given and compared to the service framework in this thesis. It should be noted that the related works of the papers in Part II and their corresponding service framework contributions are already given in every paper.

6.1 Autonomic systems and communications

General

Autonomic systems and communications are technologies inspired by the autonomic computing paradigm from IBM Research [Hor01]. Autonomic systems have the ability to manage themselves and to adapt dynamically to changes in accordance with given objectives [KC03][WHW04]. The ultimate goal of autonomic communications, however, focuses on the ability of networks and its associated devices and services to work in a totally unsupervised manner (i.e., able to self-configure, self-monitor, self-adapt, self-heal, and so on) [DDF06].

Autonomic systems are characterised by the following self-management properties: self-configuration, self-optimisation, self-healing, and self-protection [KC03][IBM06]. *Self-configuration* means that a system can install and set itself up according to high-level goals that specify what is desired, but not necessarily how to accomplish it. *Self-optimisation* means that a system optimises its use of resources. It may decide to initiate a change to the system in an attempt to improve performance or the quality of provided services. *Self-healing* means that the system can detect and diagnose problems. If possible, the system attempts to fix the problem, for example, by switching to a redundant component or by downloading and installing software updates. Fault-tolerance is an important aspect of self-healing; this involves an autonomic system being reactive to failures or early signs of a possible failure. *Self-protection* means that a system can protect itself from malicious attacks and from the end users who inadvertently make software changes. The system thus autonomously tunes itself to achieve security, privacy, and data protection. According to the above definitions of self-management, the autonomic systems based on the AEs in this thesis (see contributions **C4** and **C5**) have self-configuration and self-optimisation. Self-healing is only present in the event of node and capability failure. The autonomic systems in this thesis, however, do not have self-protection.

An autonomic system consists of interacting autonomic elements (AE). The closed feedback loop is the basic function of the autonomic system; this is the same as the adaptable service system. The closed feedback loop is usually implemented either within an AE itself or among them, based on other existing mechanisms and technologies, such as policy-based management [BRT04], intelligent agents [ZSM04], biological algorithms [SS05], context-awareness, and semantic annotations [SSS06].

Autonomic-based solutions in other research papers

Five papers [VP07][LP06][JS06][SHM09][GGZ06] applied the considered autonomic computing principles. In each of these papers, the general architectures of autonomic systems and AEs are proposed.

Vassev and Paquet [VP07] presented a generic architecture of AEs based on the Autonomic System Specification Language (ASSL) framework. Their AE specification involves service level objectives, self-management policies, friends (i.e., other AEs), interaction protocol, crash-recovery protocol, behaviour models, actions, events, and metric space.

Liu and Parashar [LP06] described the Accord programming framework, which enables the development of AEs and the formulation of autonomic systems as the dynamic composition of AEs. Here, an AE is defined by three types of ports: functional port, control port, and operational port. Rules, which incorporate high-level guidance and practical human knowledge in the form of conditional if-then expressions, are used to manage the runtime behaviours of the AE and control the interactions between AEs.

Jarrett and Seviora [JS06] described an autonomic computing architecture and its accompanying implementation infrastructure constructed on top of the Cognitive Agent Architecture (Cougaar) based on Java and XML. The functionality of the Cougaar agent, which is a dynamic mobile autonomous program, is defined by a set of plugin Java classes. The state of an agent is kept on a blackboard that is shared between all plugins running in the agent; thus, the agent can be relocated to a new node by transferring a saved snapshot of the blackboard. Based on Cougaar, an AE consists of an unspecified managed element that is governed by a Cougaar agent.

In [SHM09], the authors described the design of an AE for the FOCALE autonomic architecture. FOCALE provides a context-aware policy management approach for governing the functionality offered by the system. In this architecture, the DEN-ng information model and DENON-ng ontology are applied to translate disparate observed data into a common networking terminology. A FOCALE AE can be viewed as a programmable entity that provides functions to realise its own goals or those of the community of which it is a member. DEN-ng management policies used by the AEs define actions to both maintain the current state of the policy target and to transition the target to a new desired state.

Guo et al. [GGZ06] proposed a self-organised model of agent-enabling autonomic computing for a grid environment. Here, an AE is defined by an autonomic manager; that is, here, an AE includes an agent, managed elements, and internal relationships including recipes, interfaces, and capability and interest models. In this paper, capability represents what tasks an AE is able to process, while interest indicates the information that an AE is concerned about. AEs dynamically establish relationships to acquire each other's capabilities and interests; based on this, they can collaborate and contribute their capabilities to accomplish management tasks.

Similar to the autonomic systems based on AEs presented in this thesis (see contributions **C4** and **C5**), the other approaches [VP07][LP06][JS06][SHM09][GGZ06] provide flexibility concerning the runtime updating of the AE specification. However, this thesis's approach supports the deployment of only "incremental" changes in terms of the EFSM, policy, and/or goal specification in addition to the deployment of the whole AE specification. This feature is not mentioned explicitly in the other listed works. Likewise, the framework in [JS06] supports the movement of AEs. All other approaches also support the centralised, hierarchical semi-centralised, and decentralised autonomic system structural styles.

6.2 Policy-based network management

General

Policy-based network management (PBNM) has been an important approach for automating configurable management functionalities. It has been applied in different domains such as network security and QoS provisioning. The IETF has defined a standard framework for the PBNM system [YPG00], which essentially consists of user interfaces for creating policies, repositories for storing policies, policy decision points (PDP) for evaluating policies, and policy enforment points (PEP) where policies are enforced. The framework also defines the Common Open Policy Service (COPS) [CSD01] as the protocol for policy exchange between the PDP and PEP. The Lightweight Directory Access Protocol (LDAP) is recommended for repository access, even though other protocols such as HTTP, SNMP, or SSH are possible.

In addition, the IETF Policy Framework WG has developed the PCIM (Policy Core Information model) [MES01] as an extension of the CIM (Core Information Model) from the DMTF [DMT12]. PCIM is an object-oriented information model that defines a generic strategy for representing policies. However, some problems with this standardised information model have been revealed, such as the absence of the definition of high-level policies, and the lack of means to trigger policies [Str03]. Accordingly, several policy-based frameworks and information models for the policies have been developed. These information models are, for example, PONDER [DDL01], Rei [Kag02], and DEN-ng [Str03].

With reference to the IETF policy framework mentioned above, the RM- and AE-based service components in this thesis correspond to the PDP, whereas the EFSM-, RM-, and AE-based service components and agents are the PEP. Both CIM and the goal-based policy ontology are used the foundation for the policy specification. However, the use of the ontology can provide more powerful policy-based reasoning as explained in **C2** of Section 3.

Policy-based solutions in other research papers

Five papers [XX06][TG06][WHS08][CV06][UBL08] are briefly presented in this section.

Xiao and Xu [XX06] demonstrated the integration of ontologies into the IETF policy framework, because ontologies are able to more sufficiently express rule-based concepts as well as their existences or value dependences. The authors of this paper proposed to convert the IETF Policy information base (PIB) in ASN.1 to OWL and SWRL, and to store the converted policies in the ordinary policy repository of the policy framework.

Tsoumas and Gritzalis [TG06] gave a security management framework for an arbitrary information system that builds upon a security ontology. CIM from the DMTF is used as the basis of this security ontology, and is transformed into the ontology concepts in OWL. The security requirements and the rule-based security actions in the PONDER policy language are defined based on the ontology concepts. In addition, the security actions are high-level refined security policy statements that are to be deployed on the devices.

Wang et al. [WHS08] described a markov decision process-based multi-agent framework for the optimisation of aeroengine maintenance policies. Having a proper maintenance policy is an effective way to prolong the aeroengine. In this paper, maintenance ontology is presented. It defines the main elements that agents use to create the content of communication messages (e.g., maintenance actions, risk selection scenarios, reward types, and maintenance costs for different maintenance actions in different states).

In [CV06], the authors proposed an architecture for distributed network management tasks based on the NETCONF protocol. This system makes use of policies for defining management operations. In this paper, the policies are expressed as condition-action rules by XML, and the Distributed Manager is introduced. It mediates between the ordinary manager and agent, and is capable of taking over management operations for its network domain according to predefined policies.

Uszok et al. [UBL08] presented a generic KAoS policy service architecture and core policy ontology, which can be applied to a variety of application domains. The KAoS architecture consists of functionalities for three layers: the human interface layer for policy specification, the policy management layer for reasoning, and the policy monitoring and enforcement layer. In this framework, OWL is used to encode the policies, and the authors mention that OWL subsumption mechanisms are applied for the policy-based reasoning.

Four of the five presented papers [XX06][TG06][WHS08][UBL08] used ontologies to define policies and related information in their frameworks, which is the same as the approach in this thesis. However, only the work in [WHS08] covers the policy-based reasoning aspect. SWRL used in [XX06] and PONDER in [TG06] are analogous to how OWL/XDD is used in this thesis.

7 Conclusions and further work

7.1 Conclusions

This thesis aims to meet the following research objectives: "*to specify, construct, validate, and evaluate a service framework for capability-related adaptation in adaptable service systems.*" The service framework presented in this thesis implements the problem statements (**P1**–**P5**) defined in Section 2.3. The discussion of how each problem statement was realised by the various service framework contributions (**C1**–**C6**) was presented in Section 4.3. These service framework contributions have been included in TAPAS.

To conclude, the service framework in this thesis is composed of:

- *Generic computing architecture, concepts, and EFSM-, RM-, and AE-based service component models and mechanisms* as solid ground for the specification and execution of any adaptable service system as well as the realisation of the required functionalities. The policy-based reasoning and action rewarding mechanism in this thesis can even be applied to other research, such as context awareness and cognitive robotics.

- *Service execution platforms* for the implementation of adaptable service systems.

- *Ontologies* that define necessary concepts for capability-related adaptation, and are also open to the *integration* and the *interoperability* with ontologies in other domains.

- *System architectures* including *specific* functionalities, interworkings, and structural organisation of service components for capability-related adaptation .

Based on the generic parts of the service framework and the ontologies, five types of the *centralised* and *decentralised* closed feedback loop systems can be constructed. Each of these provides different flexibility and effectiveness, as summarised in Section 4.2.

The proposed service framework provides the following features:

- *Adaptability* in the event of node and capability failure or overload, which results in reduced system performance,

- *Flexibility*, meaning that the partial or full update of service functionalities is possible during runtime in terms of the platform software and service component specifications, and

- *Reasoning* ability to automate the functionalities for capability-related adaptation. The reasoning ability is more powerful when the ontologies are used.

7.2 Suggestions of further work

Even though the service framework presented in this thesis is quite encouraging, some issues have been left unconsidered.

Firstly, ontology verification should be considered to guarantee the correctness and consistency of the concepts in the proposed ontologies. Correctness means the concepts are lexically and syntactically correct, and consistency means that it is impossible to get contradictory conclusions from the defined concepts. Efforts are needed to study and propose mechanisms and tools to verify the proposed ontologies in this thesis as well as the requirements, policies, and goal specifications based on these ontologies. In addition, the integration of the proposed ontologies with a service ontology or a SLA ontology should be considered.

Concerning the Autonomic Element, the proposed specifications and mechanisms made self-configuration, self-optimisation, and a part of self-healing possible. Thus, future research should investigate how to support complete self-healing and self-protection in the framework. For example, an AE should ideally authenticate all requests from other AEs, and some security protocols should be applied.

Moreover, the study of the behaviors of the service framework in a large domain of available nodes and various capability types and parameters is needed. Other scenarios and experiments are also needed for more extensive evaluation. These efforts will certainly increase the level of confidence in the proposed service framework.

Finally, the service execution platforms are not fully implemented yet. Thus, further work is needed for their implementation. In addition, these service execution platforms should be published as an open source project that allows others to use and contribute to them.

Bibliography

[Aag01] F. A. Aagesen, B. E. Helvik, U. Johansen, and H. Meling, "Plug and Play for Telecommunication Functionality – Architecture and Demonstration Issues," in *Proc. of the Int. Conference on Information Technology for the New Millennium (IConIT)*, Thailand, May 2001.

[ASM98] K. Akama, T. Shimitsu, and E. Miyamoto, "Solving Problems by Equivalent Transformation of Declarative Programs," *J. Jpn. Soc. Artif. Int.*, vol. 13, pp. 944–952, 1998.

[BDD99] H. Berndt, E. Darmois, F. Dupuy, et al, *The TINA Book: A Co-operative Solution for a Competitive World*. Prentice Hall, 1999.

[Bjö10] M. Björklund, "YANG – A Data Modeling Language for the Network Configuration Protocol (NETCONF)," RFC 6020, Oct. 2010.

[BRT04] N. Badr, D. Reilly, and A. Taleb-Bendiab, "Policy-based autonomic control service – Policies for Distributed Systems and Networks," in Proc. 5th IEEE Int. Workshop on POLICY, New York, Jun. 2004, pp. 99–102.

[CDK01] G. Couloris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*. Addison-Wesley, 2001.

[CSD01] K. Chan, J. Seligson, D. Durham, S. Gai, et al, "COPS Usage for Policy Provisioning (COPS-PR)," IETF RFC 3084, Mar. 2001.

[CV06] N. Carrilho and N. Ventura, "Distributed Policy-based Network Management with NETCONF," in *Proc. Southern African Telecommunications and Applications Conference (SATNAC)*, Sep. 2006.

[DDF06] S. Dobson, S. Denazis, A. Fernandez, D. Gaıti, et al, "A Survey of Autonomic Communications," *ACM Trans. Auton. Adap.*, vol. 1, no. 2, pp. 223–259, Dec. 2006.

[DDL01] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The PONDER Policy Specification Language," *Workshop Policies for Distributed Systems and Networks*, Bristol, UK, Jan. 2001, pp. 18–39.

[DDP11] R. Di Cosmo, D. Di Ruscioa, P. Pelliccionea, A. Pierantonioa, et al, "Supporting Software Evolution – Component-Based FOSS Systems," *Sci. Comput. Program.*, vol. 76, no. 12, pp. 1144–1160, Dec. 2011.

[DHP05] Y. Diao, J. L. Hellerstein, S. Parekh, R. Griffith, et al, "A Control Theory Foundation for Self-Managing Computing Systems," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 12, pp. 2213–2222, Dec. 2005.

[DMT08] DMTF. *Web Services for Management (WS-Management) Specification*, Feb. 2008. Available: http://www.dmtf.org/standards/wsman [Last accessed April 2012].

[DMT12] DMTF. "Common Information Model (CIM)," CIM Schema Version 2.32.0, 2012. Available: http://www.dmtf.org/standards/cim [Last accessed April 2012].

[Enn06] R. Enns, "NETCONF Configuration Protocol," RFC 4741, Dec. 2006.

[GER11] K. Geihs, C. Evers, R. Reichle, M. Wagner, et al, "Development Support for QoS-Aware Service-Adaptation in Ubiquitous Computing Applications," in *Proc. 2011 ACM Symp. Applied Computing*, TaiChung, Taiwan, Mar. 2011.

[GFC03] A. Gómez-Pérez, M. Fernández-López, and O. Corcho, *Ontological Engineering*. Springer-Verlag, 2003.

[GGZ06] H. Guo, J. Gao, P. Zhu, and F. Zhang, "A Self-Organised Model of Agent-Enabling Autonomic Computing for Grid Environment," in *Proc. 6th World Congr. Intelligent Control and Automation*, China, Jun. 2006.

[HDS08] I. Hochstatter, G. Dreo, M. Serrano, J. Serrat, et al, "An Architecture for Context-driven Self-management of Services," in *Proc. IEEE Conf. Computer Communications Workshops (INFOCOM)*, USA, Apr. 2008.

[Hor01] P. Horn, "Autonomic Computing: IBM Perspective on the State of Information Technology," T. J. Watson Lab., IBM Corporation, Tech. Rep., Oct. 2001.

[IBM06] IBM Corporation, *An Architectural Blueprint for Autonomic Computing.* IBM White paper, 4th ed. Jun. 2006.

[ITU92-1] ITU-T Rec. Q1201, "Principles of intelligent network architecture," Oct. 1992.

[ITU92-2] ITU-T Rec. M.3100, "Generic Network Information Model," 1992.

[Jia08] S. Jiang, "Some Service Issues in Adaptable Service Systems," Ph.D. dissertation, Dept. Telematics, NTNU, 2008.

[Jos07] N. Josuttis, *SOA in Practice: The Art of Distributed System Design*. Sebastopol, CA: O'Reilly Media, 2007.

[JS06] M. Jarrett and R. Seviora, "Constructing an Autonomic Computing Infrastructure Using Cougaar," in *Proc. 3rd IEEE Int. Workshop Engineering of Autonomic and Autonomous Systems (EASe 2006)*, Mar. 2006, pp. 119–128.

[Kag02] L. Kagal, "Rei: A Policy Language for the Me-Centric Project," HP Labs Tech. Rep., HPL-2002-270, 2002.

[KC03] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Comput.*, pp.41–50, Jan. 2003.

[Lil11] P. Liljeback, "Modelling, Development, and Control of Snake Robots," Ph.D. dissertation, NTNU, 2011.

[LP06] H. Liu and M. Parashar, "Accord: A Programming Framework for Autonomic Applications," *IEEE Trans. Syst. Man Cybern.*, vol. 36, no. 3, pp. 341–352, 2006.

[Lüh04] E. Lühr, "Mobility Support for Wireless Devices within the TAPAS platform," M.Sc. thesis, Dept. Telematics, NTNU, 2004.

[Mcc99] K. McCloghrie et al, RFC 2578, Structure of Management Information Version 2 (SMIv2), April 1999.

[MES01] B. Moore, E. Ellesson, J. Strassner, and A. Westerinern, *Policy Core Information Model, Version 1 Specification*. IETF RFC 3060, Feb. 2001.

[MKL02] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, et al, "Peer-to-Peer Computing," in *Proc. 2nd Int. Conf. on Peer-to-Peer Computing*, Jul. 2002, pp. 1–51.

[MK00] K. McCloghrie and F. Kastenholz, *The Interfaces Group MIB*, RFC 2863, Jun. 2000.

[Nis08] O. Nistad, "SNMP-based Monitoring Application by Using TAPAS platform," M.Sc. thesis, Dept. Telematics, NTNU, 2008.

[SBF98] R. Studer, V. R. Benjamins, and D. Fensel, "Knowledge Engineering: principles and methods," Data Knowl. Eng., vol. 25, pp. 161–197, 1998.

[Shi05] M. M. Shiaa, "Mobility Management in Adaptable Service Systems," Ph.D dissertation, Dept. Telematics, NTNU, 2005.

[SHM09] J. Strassner, J. W.-K. Hong, and S. van der Meer, "The Design of an Autonomic Element for Managing Emerging Networks and Services," in *Proc. 1st IEEE Int. Conf. Ultra Modern Telecommunications & Workshops*, Oct. 2009.

[SRS07] J. Strassner, D. Raymer, and S. Samudrala, "Providing Seamless Mobility using the FOCALE Autonomic Architecture," in *Proc. 7th Next Generation Teletraffic and Wired/Wireless Advanced Networking Conference*, St. Petersburg, Russia, Sep. 2007.

[SS05] J. Suzuki and T. Suda, "A Middleware Platform for a Biologically Inspired Network Architecture Supporting Autonomous and Adaptive Applications," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 2, pp. 249–260, Feb. 2005.

[SSS06] J. M. Serrano, J. Serrat, J. Strassner, and R. Carroll, "Policy-based Management and Context Modelling Contributions for Supporting Services in Autonomic Systems," in *Proc. 1st Int. IFIP TC6 Conf. on Autonomic Networking (AN)*, Paris, France, Sep. 2006.

[Str03] J. Strassner, *Policy-based Network Management – Solutions for the next generation.* Morgan Kaufmann Publishers, 2003.

[Sup08] P. Supadulchai, "Reasoning-based Capability Configuration Management in Adaptable Service Systems," Ph.D. dissertation, Dept. Telematics, NTNU, 2008.

[TG06] B. Tsoumas and D. Gritzalis, "Towards an Ontology-based Security Management," in *Proc. 20th Int. Conf. Advanced Information Networking and Applications (AINA)*, Vienna, Austria, Apr. 2006.

[TSS97] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A Survey of Active Network Research," IEEE Commun. Mag., vol. 35, no. 1, pp. 80–86, Jan. 1997.

[UBL08] A. Uszok, J. Bradshaw, J. Lott, M. Breedy, et al, "New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of KAoS," in *Proc. Workshop on Policies for Distributed Systems and Networks*, 2008, pp. 145–152.

[Vil04] J. J. Vila-Armengol, "Implementation of Web Services Architecture in TAPAS," M.Sc. thesis, Dept. Telematics, NTNU, 2004.

[VP07] E. Vassev and J. Paquet, "Towards an Autonomic Element Architecture for ASSL," in *Proc. 29th IEEE Int. Conf. Software Engineering/Software Engineering for Adaptive and Self-managing Systems*, USA, May 2007.

[W3C04] W3C. *OWL Web Ontology Language Overview*, 2004. Available: http://www.w3.org/TR/owl-features/ [Last accessed April 2012].

[WG00] S. Waldbusser and P. Grillo, "Host Resource MIB," RFC 2790, 2000. Available: http://www.ietf.org/rfc/rfc2790.txt [Last accessed April 2012].

[WHS08] J. Wang, S. Hou, Y. Su, J. Du, et al, "Markov Decision Process Based Multiagent System Applied to Aeroengine Maintenance Policy Optimisation," in Procof the 5th Int. Conf. Fuzzy Systems and Knowledge Discovery, 2008, pp. 401–408.

[WHW04] S. White, J. Hanson, I. Whalley, D. Chess, et al, "An Architectural Approach to Autonomic Computing," in *Proc. 1st IEEE Int. Conf. Autonomic Computing*, New York, May 2004, pp. 2–9.

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

[WY04] V. Wuwonse and M. Yoshikawa, "Towards a Language for Metadata Schemas for Interoperability," in *Proc. 4th Int. Conf. on Dublin Core and Metadata Applications*, China, 2004.

[XX06] D. Xiao and H. Xu, "An Integration of Ontology-based and Policy-based Network Management for Automation," in Proc. Int. Conf. Computational Intelligence for Modelling, Control and Automation, and Int. Conf. on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06), Sydney, Australia, Nov. 2006.

[YPG00] R. Yavatkar, D. Pendarakis, and R. Guerin, "A Framework for Policy-based Admission Control," IETF RFC 2753, Jan. 2000.

[YUM11] YANG-based Unified Modular Automation toolkit for the NETCONF protocol, Feb. 2011. Available: http://www.netconfcentral.org/yuma [Last accessed April 2012].

[ZSM04] Y. Zhang, J. Sun, and H. Ma, "Self-management Model based on Multiagent and Worm Techniques," in *Proc. Canadian Conference on Electrical and Computer Engineering*, Niagara Falls, Ontario, Canada, 2004.

Part II – Research papers

Paper A: Ontology-based Capability Self-Configuration

Patcharee Thongtra, Finn Arve Aagesen and Natenapa Sriharee

Published in
Proceedings of the 14th Eunice Open European Summer School on which networks for which services

Brest, France, September 2008

Ontology-based Capability Self-Configuration

Patcharee Thongtra, Finn Arve Aagesen and Natenapa Sriharee

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

Abstract—Capability configuration management is the allocation, re-allocation and de-allocation of capabilities. A capability is here defined as an inherent physical property of a network node which is a basis for implementing networked services. Capability configuration management requires a specification of the capability configuration management process. This paper proposes a framework for capability configuration process specification (CCPS) production and execution. CCPS is a set of actions with related parameters and is used to configure the capability objects of the network nodes prior to service deployment and instantiation. A CCPS is based on Web services.

The production of CCPS is based on capability requirements defined for the networked services to be deployed and instantiated. The framework has an ontological reasoning ability. A case study to configure a VLAN connection using switches is also provided.

I. 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 system can be defined as the handling of the abstractions of the service component models of a service system through the service system life-cycle phases. The service life-cycle phases can be structures into

- service engineering,
- service configuration management and
- service discovery management.

Service engineering is the definition and implementation of the service component models. *Service configuration management* consists of capability configuration management, service deployment and service instantiation. *Service discovery management* is the process of exposing the services to the public, finding the services that satisfy the functional and QoS requirements, and accessing the services.

A capability [1] is an inherent property of a network node and is used as a basis to implement services. Capability configuration management comprises capability initialization, capability re-initialization and capability allocation adaptation. Capability initialization is the allocation and the arrangement of the capabilities for the service components to be distributed and instantiated. Capability re-initialization is the re-distribution and re-instantiation of the service components and capabilities when the instantiated service systems are unable to adapt satisfactorily as well as when capabilities are changed. Lastly, capability allocation adaptation is the monitoring of the performance of the executing service system, the re-allocation and the re-arrangement of the capabilities within the executing service systems.

Capability configuration management is a vital part of service configuration management and precedes service deployment and service instantiation. *Service deployment* is the introduction of new services and service components into the network nodes, while *service instantiation* is the creation of instances of the service components and making them ready for usage. In this paper, capability initialization comprises

- the production of a capability configuration process specification (CCPS) as well as
- the configuration of the capability objects of the processing nodes.

In this paper we propose a capability self configuration framework based on capability ontology. In the paper we are focusing on capability initialization. The framework is defined by

- information model,
- organization model and
- communication model

The information model defines the capability ontology, the capability requirement as well as CCPS. The capability ontology provides a formal way to describe and reason the semantics and axioms of the capability-related information. The capability ontology model is represented by OWL (Ontology Web Language) [2]. It expresses general capabilities of the network nodes and capability requirements of specific service system. Both are inputs to the CCPS framework. CCPS is here a set of SOAP (Simple Object Access Protocol) messages encapsulating the capability-related information to configure the capability objects of the network nodes.

The organization model defines the active components that participate in the capability configuration management; it has information model objects as input and output.

Lastly, the communication model describes SOAP messages to communicate between the active components and the network nodes. The communication model is based on WS-Management [3] which promotes the vendor-independent network management protocol by using Web services technology.

The framework described in this paper has three contributions as follows: First, the framework can automatically generate CCPS. Second, with the capability ontology the framework has a potential to reason the capability-related information in the service system. Reasoning ability is needed for a self-management service system. Third, action for the configuration is extensible. Based on Web services the network node can publish new services for the configuration without the reengineering of the framework.

The paper is organized as follows: Section II discusses related work, Section III explains the information model, and Section IV presents the organization model and the active components residing in the model. Section V describes the communication model. Section VI provides a case study of capability configuration for a VLAN (Virtual Local Area Network) connection. A summary and conclusion are provided in Section VII.

II. Related work

There have been several studies related to self-configuration networks, even though most of them have not considered a unify framework with service self-configuration management ability. With such ability the framework should be able to handle network configuration management, service deployment and service instantiation. SELFCONF [4] is an architecture for the self-configuration of networks in response to changed configuration policies. A DEN-ng specification [5] has been used as the information model and LDAP protocol as the communication model. Our work can include DEN-ng as part of the information model as well. NESTOR [6] has been demonstrated also in the automation of network configuration. The NESTOR automates configuration management by means of policy scripts with access to and manipulation of respective network nodes.

With the employment of ontology that is similar to our work, Glasner and Sreedhar [7] proposed to use ontology and OWL to model and represent the configuration of an Apache server. Based on their model the information can be validated. However, the model is not applicable for networked service systems in general. Moreover, the configuration management process for IP network nodes can be modeled by using OWL, SWRL and OWL-S, as presented in [8].

III. Information Model

Several ontologies have been proposed for use as network management information models, such as DEN-ng and CIM [9]. However, those ontologies can represent only the network nodes and their properties; they lack a representation of the relationships between them. Thus, it is difficult to reason the information.

In this paper we follow the approach of using ontology to represent the capability-related information, but we also pay attention to the relationships and their semantics. Our ontology model, *capability ontology model*, implements the capability concept. We select OWL as the representation language for the capability ontology since it has a rich set of operators for describing complex constraints.

A. Capability Ontology

A capability of a network node is described by inherent properties. Those properties represent the capabilities in terms of functional capability, resources capability and data capability with the detailed description as follows:

- Functional capability represents pure software or combined software/hardware components used for performing particular tasks which they can be measured in terms of the status.
- Resources capability represents hardware components with finite capacity. For example, processing units, storage units as well as communication units are the hardware components in which finite capacity can be measured in terms of the value.
- Data capability represents the data including the interpretation, validity and life span of which depend on the context of use.

Additionally these capabilities are defined according to the capability attributes: *negotiability*, *arrangement*, *compatibility* and *reducibility*.

With the capability attributes and their logical characteristics, such as inherited, transitive and symmetric, the capability ontology expresses the relationships and constraints between the network nodes and their properties well. Figure 1 represents the capability ontology model.

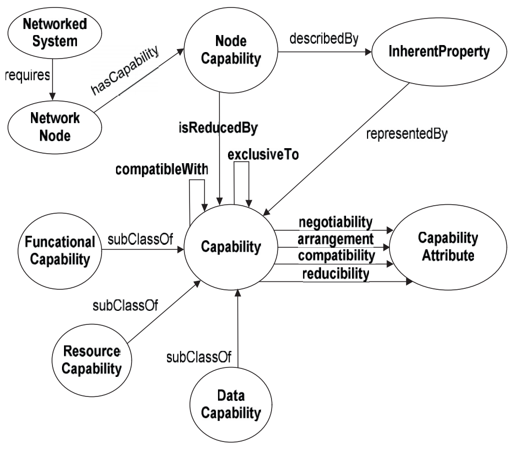


Figure 1. Capability Ontology Model.

Negotiability attribute represents whether it is possible to negotiate on a capability. The attribute is either *negotiable* or *non-negotiable*. If a capability is negotiable, there is a range of acceptable characteristics. For example, transmission channel capacity required by a text messaging application is negotiable (acceptance rate between 128 – 384 kbps), while access right data is non-negotiable.

The attribute negotiable has an inherited characteristic. If a network node capability is negotiable, then its inherent capabilities are negotiable. For example, the capability of a Web server providing normal HTTP pages is negotiable. Thus, the server's transmission channel capacity is negotiable.

Arrangement attribute represents whether a capability is limited for a special purpose (e.g., group of consumers) or it is for general use. The attribute is either *exclusive* or *shared*. For example, transmission channel and Web server process are by nature shared resources, but a password is by nature exclusive information.

Compatibility attribute represents whether a capability is definitely unique or it can be substituted by other capabilities. The attribute is either *compatible* or *incompatible*. For example, JRE (Java Runtime Environment) v.1.4.2 is compatible with JRE v.1.4.7, but not the other way round.

The attribute compatible has a transitive characteristic, while the attribute incompatible has a symmetric characteristic.

Reducibility attribute represents whether a capability is reduced proportional to the consumer (e.g. the user or other capabilities). The attribute is either *reducible* or *irreducible*. For example, bandwidth is decreased by the number of users multiplied by capacity consumed per user, while a Web server running process is non-reducible.

The attribute reducible has an inherited characteristic as well as a transitive characteristic. For example, the capability of a switch node is reducible; its memory is reduced by data in routing table.

B. Capability Requirement

The capability description describes the capability of the network node in general without specific capacity and status. The capability requirement states that which classes of the network nodes are needed in the service systems including the number of each class. It utilizes the capability description and specifies required capacity and status.

Table 1 represents an example of a VLAN streaming service system that consists of client switch and server switch. Both client switch and server switch are subclasses of a network node. The client switch capability is negotiable. The server switch is described by these capabilities: Interface, IOS (Internetwork operating system), NVRAM (Non-volatile random access memory), etc. Interface is a functional capability which is measured by its status. IOS is also a functional capability. IOS is compatible with the IOS itself. NVRAM is a reducible resource capability which has finite memory size that can be measured in terms of megabyte units. In addition, it contains administrator password that is an exclusive data capability.

Table 1. An example of the capability description for VLAN streaming service system.

```
<owl:Class rdf:ID=”VLAN_Streaming_ServSystem”>
   <rdfs:subClassOf
     rdf:resource=”#NetworkedSystem”/>
   <requires rdf:resource=”#ServerSwitch”/>
   <requires rdf:resource=”#ClientSwitch”/>
</owl:Class>
<owl:Class rdf:ID=”ClientSwitch”>
    <rdfs:subClassOf rdf:resource=”#NetworkNode”/>
    <hasCapability
rdf:resource=”#ClientSwitchCapability”/>
</owl:Class>
<owl:Class rdf:ID=”ClientSwitchCapability”>
   <negotiability rdf:resource=”#Negotiable”/>
</owl:Class>

<owl:Class rdf:ID=”ServerSwitch”>
    <rdfs:subClassOf rdf:resource=”#NetworkNode”/>
    <hasCapability
rdf:resource=”#ServerSwitchCapability”/>
</owl:Class>
<owl:Class rdf:ID=”ServerSwitchCapability”>
   <describedBy rdf:resource=”#Interface”>
   <describedBy rdf:resource=”#IOS”>
   <describedBy rdf:resource=”#NVRAM”>
   ...
</owl:Class>
<owl:Class rdf:ID=”Interface”>
   <rdfs:subClassOf
rdf:resource=”#FunctionalCapability”/>
   <hasStatus rdf:resource=”#InterfaceStatus”/>
</owl:Class>
<owl:Class rdf:ID=”IOS”>
   <rdfs:subClassOf
rdf:resource=”#FunctionalCapability”/>
   <compatibility rdf:resource=”#Compatible”>
   <compatibleWith rdf:resource=”#IOS”/>
</owl:Class>
<owl:Class rdf:ID=”NVRAM”>
   <rdfs:subClassOf
rdf:resource=”#ResourceCapability”>
   <hasFiniteCapacity rdf:resource=”#MemorySize”/>
   <hasFiniteCapacityUnit ref:resource=”#MB”/>
   <reducibility rdf:resource=”#Reducible”/>
</owl:Class>
```

```
<owl:Class rdf:ID=”AdministratorPassword”>
  <rdfs:subClassOf
    rdf:resource=”#DataCapability”>
  <arrangement rdf:resource=”#Exclusive”/>
</owl:Class>
```

From the capability description, the capability requirement can be defined further. Table 2 represents the capability requirement for VLAN streaming service system that requires two client switches and a server switch. The server switch requires at least 512MB NVRAM, installs IOS_12.1.8a, and implements VTP (Virtual Trunking Protocol). The requirement defines compatibility between two versions of acceptable IOS; it defines IOS_12.1.8a and IOS_12.1.9 to be compatible. Also, it describes an arrangement between VTP and VLAN information that VTP_1 is exclusive to VLAN_Savanne_Room (VLAN name “savanne_room” and no. 5).

Table 2. The capability requirement for VLAN streaming service system in NTNU room.

```
<VLAN_Streaming_ServSystem
rdf:ID=”NTNU_Savanne_Room”>
  <requires>
    <ClientSwitch rdf:ID=”Client_Switch_1”/>
  </requires>
  <requires>
    <ClientSwitch rdf:ID=”Client_Switch_2”/>
  </requires>
  <requires rdf:resource=”#Server_Switch_1”/>
</VLAN_Streaming_ServSystem>
<ServerSwitch rdf:ID=”Server_Switch_1”>
  <hasCapability
  rdf:resource=”#Server_Switch_Cap_1”/>
</ServerSwitch>
<ServerSwitchCapability
rdf:ID=”Server_Switch_Cap_1”>
  <describedBy rdf:resource=”#NVRAM_1”/>
  <describedBy rdf:resource=”#IOS_12.1.8a”/>
  <describedBy rdf:resource=”#VTP_1”/>
  ...
</ServerSwitchCapability>
<NVRAM rdf:ID=”NVRAM_1”>
  <hasFiniteCapacity
  rdf:resource=”#MemorySize_1”/>
</NVRAM>
<MemorySize
rdf:ID=”MemorySize_1”>512</MemorySize>
<IOS rdf:ID=”IOS_12.1.8a”>
  <compatibleWith rdf:resource=”#IOS_12.1.9”/>
</IOS>
<VTP rdf:ID=”VTP_1”>
  <exclusiveTo
  rdf:resource=”#VLAN_Savanne_Room”/>
</VTP>
<VLAN_Info rdf:ID=”VLAN_Savanne_Room”>
  <name>savanne_room</name>
  <number>5</number>
</VLAN_Info>
```

C. Capability Configuration Process Specification (CCPS)

CCPS is an XML-based specification composed of SOAP messages. The messages are sent to services implemented on the network nodes. The services are to get, update, create, delete and subscribe the capability of the network nodes. The capabilities are realized by the capabilities objects.

A message contains the configuration action (get, update, create, delete and subscribe) and the configuration action parameters. The configuration action and parameters are enclosed by the element *Action* and *ActionParams* respectively, while the configuration result is identified by the element *ActionResponse*. (Each message is detailed in Section V.) The configuration action parameter is likely a capability class (defined by the capability description). Table 3 illustrates the schema for CCPS.

Table 3. Schema for CCPS.

```
<xs:schema targetNamespace=
”http://schemas.tapas.item.ntnu.no/Capability”
xmlns:xs=”http://www.w3.org/2001/XMLSchema”
xmlns:ws=”http://schemas.xmlsoap.org/ws/2005/06/ma
nagement”
xmlns:env=”http://schemas.xmlsoap.org/soap/envelope
”
xmlns:cap=”http://schemas.tapas.item.ntnu.no/Capabili
ty”>
  <xs:import
namespace=”http://schemas.xmlsoap.org/ws/2005/06/
management”
schemaLocation=”wsmanagement.xsd”/>
  <xs:import
namespace=”http://schemas.xmlsoap.org/soap/envelop
e” schemaLocation=”soap.xsd”/>

  <xs:element name=”CCPS” type=”CCPS_Type”/>
  <xs:complexType name=”CCPS_Type”>
```

```
      <xs:sequence>
        <xs:element ref="networkSystemID"/>
        <xs:element ref="env:Envelop" minoccurs=1/>
      </xs:sequence>
   </xs:complexType>

    <xs:element name="networkSystemID"
 type="String"/>
    <xs:complexType name="Header">
      ...
      <xs:element ref="wsa:Action"/>
      <xs:element ref="ActionParams"/>
    </xs:complexType>

    <xs:complexType name="Body">
      ...
      <xs:element ref="ActionResponse"/>
    </xs:complexType>

    <xs:element name=ActionParams
 type="ActionParams_Type"/>

    <xs:complexType name="ActionParams_Type">
      <xs:sequence minoccurs=1>
        <xs:element ref="ActionParam"/>
        <xs:element ref="ActionValue"/>
      </xs:sequence>
   </xs:complexType>
    <xs:element name="ActionParam" type="String"/>
    <xs:element name="ActionValue" type="String"/>

    <xs:element name=ActionResponse type="String"/>
 </xs:schema>
```

IV. Organization model

The organization model is layered into three levels; *User Interface Layer*, *Computing Layer* and *Network Nodes Layer*, as shown in Figure 2. The computing layer consists of three components; *Capability Manager (CM)*, *Query Manager (QM)* and *Capability Object Configurator (COC)*, one repository; *Capability Ontology Repository (CORep)* and Web services registry; *Capability Web Services Registry (CWSReg)*. CORep stores the capability description, the capability requirement and the capability instance. CWSReg is where the network nodes register their Web services for the capability configuration and notifications.

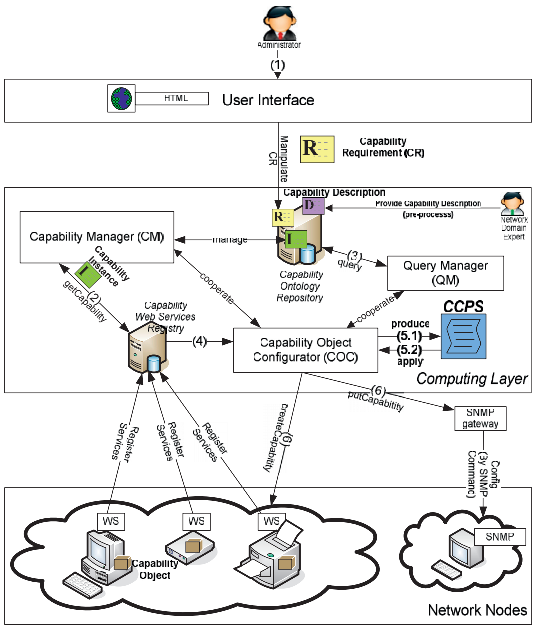


Figure 2. The organization model.

First, the administrator inputs, queries and updates the capability requirement via the user interface (1). The capability description has been provided by the network domain expert to support the creation of the capability requirement. After the capability requirement is input, the components residing in the computing layer start their processes automatically.

The CM is responsible for getting the capability instance from existing network nodes via the *getCapability* service (2). A capability instance is a data representation of a capability of the network node. In addition, the CM manages the capability instance for example providing an access to the instances.

The QM generates query commands to select (3) either the network nodes whose capabilities satisfy the capability requirement, or the ones having potential to meet the capability requirement. For example, a query to find a PC with free disk space more than 1GB. QM uses the transformation rule to transform the capability requirements into SPARQL (SPARQL Protocol and RDF Query Language)

[10] query format. (The transformation rule is given in Appendix.) The result of the query can identify the network node by its URI (Uniform Resource Identifier).

The COC gets the URIs of the selected network nodes from the QM. Then the COC finds their *setCapability*, *createCapability* and *deleteCapability* services (4) to make their capabilities in absolute compliance with the capability requirements. The COC makes a CCPS (5.1). Lastly the COC applies the CCPS (5.2) by sending the messages residing on the CCPS to the network nodes (6).

At the network nodes layer the nodes register their services to CWSReg. According to WS-Management specification, the network nodes implement Web services allowing others to manage their resource instances, which are considered here as capability objects. However, the network nodes can also implement custom services or custom actions besides the ordinary well-defined services. (For network nodes that have been implemented other network management architectures, such as SNMP, we use a software gateway for the information exchange between a protocol used in such architectures and Web services protocol.)

V. Communication model

In this paper we apply WS-Management framework which proposes Web services as the protocol for managing network systems. WS-Management exposes the network management information model - such as CIM or custom models - as Web services. Several works [11, 12, 13] analyzed the advantages and drawbacks of the employment of Web services technology for the network management compared to the employment of classical network management architectures: SNMP or CMIP. However, Web services is a current technology trend for the networks with self-management ability as mentioned in [14].

The communication model extends WS-Management. It details custom actions and SOAP messages for the communication between the computing layer and the network nodes. The custom actions are defined in order to manipulate the capability as follows:

- *getCapability action* is to request for the capability instance.
- *getCapabilityResponse action* is a response to getCapability to return the capability instance.
- *setCapability, createCapability* and *deleteCapability action* are to respectively set, create and delete the capability objects.
- *setCapabilityResponse, createCapabilityResponse* and *deleteCapabilityResponse action* are to respectively return the result of setCapability, createCapability and deleteCapability.
- *subscribeOnCapability action* is an action to subscribe event relating to any changes that occur with the capability objects. When an event occurs, an asynchronous message notification is sent to indicate that event. An event is for example low disk space.
- *subscribeOnCapabilityResponse action* is to return the result of a subscription done by subscribeOnCapability action.
- *unsubscribeOnCapability action* is to terminate a subscription unless the subscribed capability is used further ahead.
- *unsubscribeOnCapabilityResponse action* is to return the result of unsubscribeOnCapability action.

We do not need subscribeOnCapability, unsubscribeOnCapability and their response actions for the capability allocation. But we also consider them in order to support the capability re-allocation in the future.

The request actions (getCapability, setCapability, createCapability, deleteCapability, subscribeOnCapability and unsubscribeOnCapability) and their parameters are enclosed in the message template displayed in Table 4, while the response actions' message template is shown in Table 5.

Note that XML namespaces used in the templates follow WS-Management specification.

Table 4. The request action message template.

```
<env:Header>
  <wsa:To>management service AP on node</wsa:To>
    <wsman:ResourceURI>WSDL identity and port on
node</wsman:ResourceURI>
    <wsa:ReplyTo>component URI</wsa:ReplyTo>
    <wsa:MessageID>uuid:xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx</wsa:MessageID>
    <wsa:Action>action URI</wsa:Action>
    <cap:ActionParams>
      <cap:ActionParam>param<cap:ActionParam>
      <cap:ActionValue>value<cap:ActionValue>
    </cap:ActionParams>
<env:Header>
```

Table 5. The response action message template.

```
<env:Header>
  <wsa:To>
http://schemas.xmlsoap.org/ws/2004/08/addressing/rol
e/anonymous
  </wsa:To>
    <wsa:Action>action URI</wsa:Action>
<env:Header>
<env:Body>
  <cap:ActionResponse>value</cap:ActionResponse>
</env:Body>
```

In the message templates we use XML elements defined in WS-Management specification and our proposed schema as follows:

- *wsa:To* (in the request action message template) indicates the transport address of management service access point (AP) on the network node,
- *wsman:ResourceURI* denotes WSDL (Web Services Description Language) identity and port on the network node,
- *wsa:ReplyTo* denotes URI of the component in the computing layer which will get the response,
- *wsa:MessageID* represents a unique message id for tracking and correlation between the request and the response.
- *wsa:Action* identifies the action URI.
- *cap:ActionParams* encloses different parameters, that corresponds to the capability description, depending on the request action. (See an example from the case study.)
- *wsa:To* (in the response action message template) denotes the requestor.
- *cap:ActionResponse* defines successful or unsuccessful status of the request action.

VI. Case Study

The case study is to configure a VLAN connection using Cisco switches. The configuration is prepared for a streaming service system.

Switches allow a network to be partitioned into logical segments through the use of VLAN. Switches which share Layer 2 VLAN communication need to be connected by a trunk. IEEE 802.1Q and Virtual Trunking Protocol (VTP) are two popular protocols for VLAN trunks. VTP has been developed on Cisco switches to centralize the creation and deletion of VLANs in a network into a VTP server switch. The server switch has VTP mode server. It can take care of creating, deleting and updating the status of existing VLANs to the other switches sharing the same VTP domain.

This example requires three switches: one server switch and two client switches, to implement a VLAN connection in the NTNU Savanne Room. Figure 3 depicts switches and their properties in the room before the configuration.

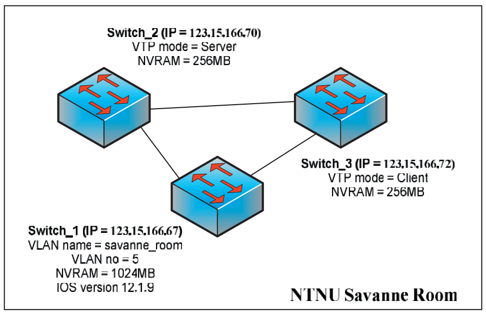


Figure 3. Existing switches in NTNU Savanne Room.

A. The capability description and the capability requirement

Figure 4 depicts the capability description of the server switch that is considered in the example. In additional to the capabilities already described in Table 1 the server switch implements VTP protocol and stores VLAN information. VTP concerns VTP domain and VTP mode. VLAN information consists of a VLAN name and a VLAN number. The VLAN information is stored in NVRAM.

The capability requirement of this example is expressed in Table 2.

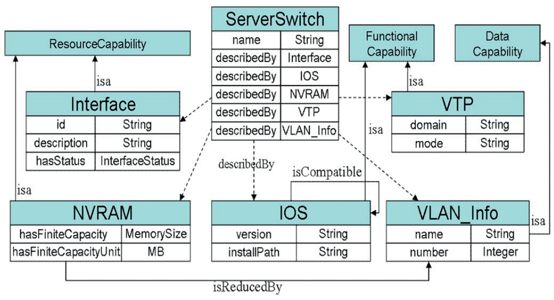


Figure 4. Server Switch Capability Description.

B. The query command

The following shows a query command which is generated to query for the server switch. The query means we need a ServerSwitch with minimum 512MB NVRAM. It must install IOS that is compatible with IOS 12.1.8a, and is reserved for VLAN in the Savanne Room. (See Appendix for details.)

We only consider the server switch as it is non-negotiable, while the client switch is negotiable. Moreover we pay attention to the VLAN information of the server switch as it will be distributed to other client switches automatically.

From the Figure 3 the query returns switch_1 as the ServerSwitch.

```
Select ?server_instance
Where
?server_instance rdf:type ServerSwitch .
?server_instance hasCapability $server_cap .
?server_cap describedBy ?nvram_instance .
?nvram_instance
   hasFiniteCapability ?memory_size .
FILTER (?memory_size >= 512) .

?server_cap describedBy ?vtp_instance .
?vtp_instance
  exclusiveTo VLAN_Savanne_Room .

?server_cap describedBy ?ios_instance .
IOS_12.1.8a compatibleWith ?ios_instance
```

C. CCPS

The following shows a CCPS of this example. It consists on two messages. The first message is to set switch_1 as server, and the second message is to set switch_2 as client.

```
<CCPS>
   <networkSystemID>NTNU_Savanne_Room</
networkSystemID>

   <!-- The first message -->
   <env:Envelop>
     <env:Header>
       <wsa:To>http://123.15.166.67/wsman</wsa:To>
     <wsman:ResourceURI>
http://schemas.tapas.item.ntnu.no/2008/04/Switch/
     </wsman:ResourceURI>
       <wsa:ReplyTo>
http://schemas.xmlsoap.org/ws/2004/08/addressing/role/an
onymous
     </wsa:ReplyTo>
       <wsa:MessageID>uuid:d9726315-bc91-430b-9ed8-
ce5ffb858a87</wsa:MessageID>
       <wsa:Action>
http://actions.tapas.item.ntnu.no/2008/04/Setcapability
       </wsa:Action>
       <cap:ActionParams>
         <cap:ActionParam>
           VTP/mode
         </cap:ActionParam>
         <cap:ActionValue>server</cap:ActionValue>
       </cap:ActionParams>
     </env:Header>
   </env:Envelop>

   <!-- The second message -->
   <env:Envelop>
     <env:Header>
       <wsa:To>http://123.15.166.70/wsman</wsa:To>
       <cap:ActionParams>
         <cap:ActionParam>
           VTP/mode
         </cap:ActionParam>
```

```
      <cap:ActionValue>client</cap:ActionValue>
    </cap:ActionParams>
    ...
  </env:Header>
 </env:Envelop>
</CCPS>
```

VII. Conclusion

A capability self-configuration framework for networked service systems has been developed, and a case study to configure VLAN connection has been demonstrated. In the framework, there are computing components to get the capability requirement, to search for the network nodes with potential capabilities, and to compute and execute a specification CCPS automatically. The capability configuration management information needed is modeled by the capability ontology. The capability attribute is a part of the proposed ontology that supports the reasoning of the information. The configuration actions in CCPS based on Web services consist of the action get, set, create, delete as well as subscribe on the capabilities.

The framework also provides a potential to support the modeling of service deployment and service instantiation in addition to the capability allocation that has been the focus in this paper. Extension of the capability ontology to support the modeling of service level agreements (SLA) specification based on the capability is also considered.

Appendix

In this section the transformation rule for the finite capacity capability and the capability attribute are provided.

Rule 1 (for finite capacity capability)

If a node is described by a finite capacity capability (expressed by <hasFiniteCapacity> element), the following query is generated to select the node's instance that has such capability with minimum defined value.

```
Select ?node_instance
Where
?node_instance rdf:type node .
?node_instance hasCapability $node_cap .
?node_cap describedBy ?capability_instance .
?capability_instance
  hasFiniteCapability ?finite_capability .
FILTER (?finite_capability >= capacity_value)
```

For example, from the Table 2 a query is generated to select a server switch that has memory size >= 512MB as follows:

```
Select ?server_instance
Where
?server_instance rdf:type ServerSwitch .
?server_instance hasCapability $server_cap .
?server_cap describedBy ?nvram_instance .
?nvram_instance
   hasFiniteCapability ?memory_size .
FILTER (?memory_size >= 512)
```

Rule 2 (for arrangement attribute)

If a node is described by an excusive capability (expressed by <arrangement rdf:resource="#Exclusive"/>), the following query is generated to select its instance, when there is a requirement that the capability is exclusive to another capability.

```
Select ?node_instance
Where
?node_instance rdf:type node .
?node_instance hasCapability $node_cap .
?node_cap describedBy ?capability_instance .
?capability_instance
  exclusiveTo another_capability
```

For example, from the Table 2 a query is generated to select a server switch that is exclusive to VLAN defined for Savanne Room as follows:

```
Select ?server_instance
Where
?server_instance rdf:type ServerSwitch .
?server_instance hasCapability $server_cap .
?server_cap describedBy ?vtp_instance .
?vtp_instance
  exclusiveTo VLAN_Savanne_Room
```

Rule 3 (for compatibility attribute)

If a node is described by a compatible capability (expressed by <compatibility rdf:resource="#Compatible"/>), the following query is generated to select its instance.

```
Select ?node_instance
Where
?node_instance rdf:type node .
?node_instance hasCapability $node_cap .
?node_cap describedBy ?cap_instance .
compatible_cap compatibleWith ?cap_instance
```

For example, from the Table 2 a query is generated to select a server switch that installs IOS that is compatible with IOS 12.1.8a as follows:

```
Select ?server_instance
Where
?server_instance rdf:type ServerSwitch .
?server_instance hasCapability $server_cap .
?server_cap describedBy ?ios_instance .
IOS_12.1.8a compatibleWith ?ios_instance
```

References

[1] F. A. Aagesen, and P. Supadulchai, "A Capability-based Service Framework for Adaptable Service Systems," in *Proc. of 2nd Int. Conf. on Advances in Information Technology (IAIT2007)*, Thailand, Nov. 2007.

[2] *Web Ontology Language*, W3C, 2004. Available: http://www.w3.org/2004/OWL

[3] *Web Services for Management (WS-Management) Specification*, DMTF, Feb. 2008. Available: http://www.dmtf.org/standards/wsman

[4] R. Boutaba, S. Omari, and A. P. S. Virk, "SELFCON: An Architecture for Self-Configuration of Networks," in *Journal of Communications and Networks*, Vol. 3, No. 4, Dec. 2001.

[5] *Directory Enabled Network (DEN) initiative*, DMTF. Available: http://www.dmtf.org/standards/wbem/den

[6] Y. Yemini, A. V. Konstantinou, and D. Florissi, "NESTOR: An Architecture for Network Self-Management and Organization," in *IEEE Journal on Selected Areas in Communications*, Vol.18, No.5, pp.758-766, May 2000.

[7] D. Glasner, and V. C. Sreedhar, "Configuration Reasoning and Ontology For Web," in *IEEE Conf. on Service Computing (SCC 2007)*, Jul. 2007.

[8] X. Hui, and X. Debao, "A Common Ontology-Based Intelligent Configuration Management Model for IP Network Devices," in *Proc. of 1st Int. Conf. on Innovative Computing, Information and Control, 2006 (ICICIC 06)*, Aug. 2006.

[9] *Common Information Model (CIM) Standard*, DMTF. Available: http://www.dmtf.org/standards/cim

[10] *SPARQL query language for RDF*, W3C, Jan. 2008. Available: http://www.w3.org/TR/rdf-sparql-query

[11] A. Pras, T. Drevers, R. van de Meent, and D. Quartel, "Comparing the Performance of SNMP and Web Services-Based Management," in *eTransactions on Network and Service Management 1(2)*, Dec. 2004.

[12] G. Pavlou, P. Flegkas, S. Gouveris, and A. Liotta, "On Management Technologies and the Potential of Web Services," in *IEEE Communications Magazine 42(7)*, Jul. 2004.

[13] R. L. Vianna, M. J. B. Almeida, L. M. R. Tarouco, and L. Z. Granville, "Investigating Web Services Composition Applied to Network Management", in *Int. Conf. on Web Services, 2006 (ICWS'06)*, Sep. 2006.

[14] J. Schönwälder, A. Pras, and J. P. Martin-Flatin, "On the Future of Internet Management Technologies," in *IEEE Communications Magazine*, Vol. 41, Issue 10 (2003), pp.90-97, Oct. 2003.

Paper B: Capability Ontology in Adaptable Service System Framework

Patcharee Thongtra and Finn Arve Aagesen

Published in
Proceedings of the IARIA/IEEE 5th International Multi-Conference Computing in the Global Information Technology (ICCGI 2010)

Valencia, Spain, September 2010

Capability Ontology in Adaptable Service System Framework

Patcharee Thongtra
Department of Telematics
Norwegian University of Science and Technology
N-7491 Trondheim, Norway
patt@item.ntnu.no

Finn Arve Aagesen
Department of Telematics
Norwegian University of Science and Technology
N-7491 Trondheim, Norway
finnarve@item.ntnu.no

Abstract—This paper presents a Capability Ontology (CapOnt) and a rule-based reasoning mechanism, which support service management within adaptable service systems. The ontology concepts comprise capability types, capability parameters and service management functions related to capabilities. Capability parameter values can be defined by constraints on other capability parameters. The service management functions included in the ontology are Capability Administration, Capability Configuration, Capability Allocation and Capability Performance Diagnosis. The service management functions are defined by rules consisting of constraints and management actions. The ontology concepts are represented in OWL (Web Ontology Language) and OWL/XDD (XML Declarative Description Language) – a rule-oriented knowledge representation. An intelligent conference room example with simulation results, that demonstrates the CapOnt and the rule-based reasoning, is also presented.

Keywords-Capability; Network and service management; Adaptable system; Ontology

I. Introduction

In this paper, networked services are considered. A service system consists of inter-working service components. An adaptable service system is here defined as a service system that can adapt dynamically to changes to service users, nodes, capabilities, system performance and service functionality. *A capability is an inherent property of a node to implement service [1]*. Capabilities can be classified according to capability types and capability parameters. A network interface with finite bandwidth is an example of a capability type. A service component needs capabilities to be allocated before deployment and instantiation. The capabilities can be re-allocated in situations when the system performance is not satisfactory. Formal concepts are, therefore, needed as a basis for the administration, configuration, allocation, monitoring and performance diagnosis of capabilities.

The software mechanisms used for implementing the functionality of adaptable service systems must be flexible and powerful. Service components based on the classical Extended Finite State Machine (EFSM) approach can be flexibly executed by using generic software components that can download and execute different EFSM-based specifications. In addition, combining *rule-based reasoning* with the EFSM-based approach adds the ability to cope with various situations more flexible. The rule-based reasoning is considered as a support functionality for the EFSM-based service components; it suggests actions to the service components.

Considering the capability concepts, there are some standard information models comprising defined capabilities, e.g. SNMP's Structure of Management Information (SMI) [2] and Common Information Model (CIM) [3]. However, they are limited in terms of using rule-based concepts. In this paper, we propose an ontological approach to model and represent the capability concepts including rule-based concepts.

An ontology is a formal and explicit specification of a *shared* conceptualization [4]. An ontology in general consists of types, properties, instances, relations and rules. The rule can place constraints on sets of types, properties and relations allowed between them. *Capability Ontology (CapOnt)* has capability types, parameters, instances, relations and service management functions. The parameters are the properties and inference relations of the capability types. The parameter values can be defined by constraints on other parameters. The service management functions, which are capability-related actions to fulfill the service management functionalities, are also included in this ontology. This is because the actions are shared concepts among distributed service components which implement the service management functionalities. The service management functions are defined by rules consisting of constraints and the actions.

In principle, all network management objects, operations on these objects, and network and service management functionality can be defined in ontologies. In this paper, our ontology defines capability types, parameters and service management functions.

The main contributions of this paper are:

- A service functionality architecture for adaptable service systems.
- A Capability Ontology (CapOnt) that includes rule-based parameters and service management functions.
- A rule-based reasoning procedure. The reasoning procedure is implemented by Native xml Equivalent Transformation reasoning engine (NxET) [5].

The rest of the paper is organized as follows. Sec. II reviews related works. Sec. III presents the service functionality architecture. Sec. IV presents the Capability Ontology (CapOnt) concepts and representation. Sec. V explains the reasoning procedure. In sec. VI, an example of intelligent conference room is given. Finally, sec. VII presents summary and conclusions.

II. Related Work

A. Standardized Information Models

The IETF proposed Structure of Management Information (SMI) [2] to be used in the SNMP framework [6], followed by Next Generation Structure of Management Information (SMIng) [7]. SMIng provides mechanisms to formally specify constraints between values of multiple parameters, though its implementation was not accomplished. In the other models: Guidelines for the Definition of Managed Objects (GDMO) [8] and Common Information Model (CIM) [3], the object-oriented modeling approach is applied. GDMO has been widely used in telecommunications networks. CIM provides the definitions for systems, networks, users and services. It was proposed with the intention to use in the Web-based management architecture. However, we can not easily define rule-based managed objects, which their values are restricted by other managed objects, by these information models alone. SNMP RMON MIB allows that to some extent for statistics gathering, but it is complicated. López de Vergara et al. [9] used OCL as an extension model to write rules in the CIM metaschema.

B. The Application of Ontology

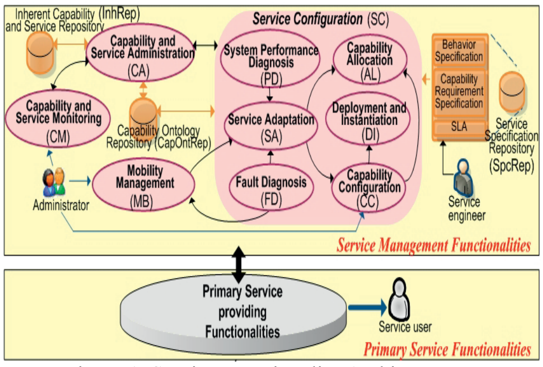
There are various directions for the application of ontology. In [10]-[12], the ontology-based mapping techniques are used to integrate existing information models. In [13]-[15], the formal definitions of concepts including rules by the ontology-based approaches were presented. Guerrero et al. [13] used OWL [16] and SWRL to define constraints on the parameter values of the SNMP- and the CIM managed objects. They also modeled behaviour rules composed of the actions and constraints. The actions are general functions, that are performed by the manager in the traditional manager-agent framework when the constraints are met. In [14], the behaviour rules were defined with a focus on the network configuration management. Diaz et al. [15] mapped the original configuration of wireless routers into the CIM schema, and later translated these into OWL. They further added constraints on the relations allowed between the managed objects, with which to detect router configuration errors. The approach used in this paper is similar to [13],[14]. But the Capability Ontology has an application domain different from the others. The capability concepts are defined towards the service management functionalities for adaptable service systems.

III. Service Functionality Architecture

Service functionality architecture as illustrated in Fig. 1 defines the structure and content of service functionalities independent of implementation. The architecture is intended to provide three classes of the adaptability: *rearrangement flexibility*, *failure robustness*, and *QoS awareness and resource control*. *Rearrangement flexibility* means that the service system structure and functionality are not fixed. Nodes, users, services, service components, capabilities, can be added, moved, removed according to needs. Mobility of users, sessions, nodes, terminals is further seamlessly handled. *Failure robustness* means that the architecture is dependable and distributed, and that the system can reconfigure itself in the presence of failures. *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. Fundamental QoS concepts considered are capability, capability performance, service performance and service level agreements (SLAs).

The functionalities of the service functionality architecture are classified as *primary service functionalities* and *service management functionalities*. The *primary service functionalities* provide services to the service users. The *service management functionalities* consist of functionality components and repositories to manage services as well as nodes and their capabilities. The functionality components and repositories will be explained later in this section.

The service management functionalities and the primary service functionalities can be implemented by Extended Finite State Machine (EFSM)-based and Reasoning Machine (RM)-based service components [17]. In this paper, the RM-based service component is used as the traditional procedure that provides the rule-based reasoning to the EFSM-based service component.


Inherent Capability (InhRep)
and Service Repository
Capability and Service Administration (CA)
Service Configuration (SC)
System Performance Diagnosis (PD)
Capability Allocation (AL)
Behavior Specification
Capability Requirement Specification
SLA
Service Specification Repository (SpcRep)
Capability and Service Monitoring (CM)
Capability Ontology Repository (CapOntRep)
Service Adaptation (SA)
Deployment and Instantiation (DI)
Administrator
Mobility Management (MB)
Fault Diagnosis (FD)
Capability Configuration (CC)
Service engineer
Service Management Functionalities
Primary Service providing Functionalities
Service user
Primary Service Functionalities
Figure 1. Service Functionality Architecture.

- *Capability Ontology Repository (CapOntRep)* stores the capability concepts.
- *Service Specification Repository (SpcRep)* stores the service component behavior specifications, the capability requirements, and the SLAs. The capability requirements define the required capability types and parameters for the various service components.
- *Inherent Capability and Service Repository (InhRep)* stores data about available nodes, the inherent capabilities and instantiated services. The inherent capabilities are the existing instances of capability types and parameters for the various nodes.
- *Capability and Service Administration (CA)* handles the registration of nodes, the inherent capabilities and instantiated services. CA also provides a current view of available nodes, the inherent capabilities and instantiated services.
- *Capability and Service Monitoring (CM)* monitors node, their capabilities and instantiated services. CM gets the

monitoring requests from the administrator as well as CA, and updates the monitored data to CA.

- *Mobility Management (MB)* handles various mobility types such as personal mobility, terminal mobility and service component mobility.
- *Service Configuration (SC)* has the sub-components as illustrated in the figure. *Capability Configuration (CC)* finds and selects capable nodes that satisfy the capability requirements. The *capable node* has capabilities as same as or as compatible as ones specified in the requirements. *Capability Allocation (AL)* allocates capabilities to service components in various service classes. The task is performed in accordance with the requirements of system performance, here defined as the sum of service performance and capability performance, in the SLAs. *Deployment and Instantiation (DI)* comprises *deployment* which is the introduction of new service components in nodes, and *instantiation* which starts execution of deployed service components. *Fault Diagnosis (FD)* detects the failure of the instantiated services. *System Performance Diagnosis (PD)* detects mismatch between the required system performance and the inherent system performance. *Service Adaptation (SA)* plans the adaptation during the service system execution in case of 1) mismatch between the required system performance and the inherent system performance, 2) faults and 3) other reasons for service component movements. The adaptation results in AL only, or the combination of CC and AL.

In this paper, the capability is focused on. Accordingly, CA and PD are simplified to Capability Administration and Capability Performance Diagnosis. AL considers only the required capability performance.

IV. Capability Ontology

A. Concepts

The concepts of the Capability Ontology (CapOnt) are illustrated in Fig. 2. *Capability type* defines entities with common characteristics. A capability type can be functions, physical resources and data. The function examples are operating system, application software and protocol. The physical resource examples are CPU and memory. User accounts and passwords are the data examples.

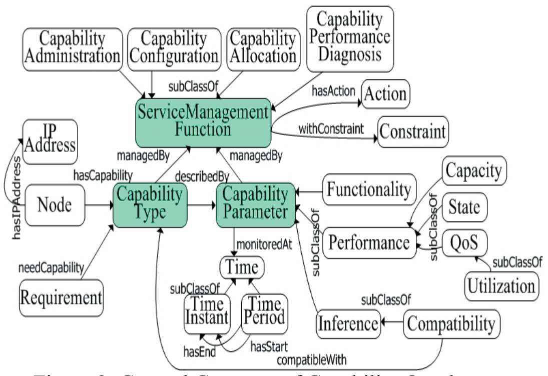


Figure 2. General Concepts of Capability Ontology.

Capability parameter describes the characteristics of a capability. *Functionality* parameters define features of the functionality and *performance* parameters describe features of the performance. *Inference* parameters define relations to other capability types, in which these relations are logically concluded from other parameters. The functionality and the performance parameters *can* be rule-based, while the inference parameters *are* rule-based.

Performance parameters are further classified into *capacity*-, *state*- and *QoS* parameters. *Capacity* parameter examples are transmission channel capacity, CPU processing speed and disc size. *State* parameters define the situation of the capability at a specific time. State parameter examples are the number of available streaming connections and number of packets discarded in a specific buffer. *QoS* parameter defines the degree of satisfaction of the service users. QoS parameters can be traffic and dependability statistics based on observed stochastic variables. Important traffic variable examples are throughput and utilization. Important dependability variable examples are availability and recovery time.

The parameter *Utilization* and *Compatibility* are important rule-based parameters used in the example in Sec. VI. *Utilization* is a QoS parameter that gives the average usage of a capacity over a defined time interval. *Compatibility* is an inference parameter that specifies a relation from a capability to other compatible capabilities, in which these capabilities can be used as its substitutes.

The service management functions are defined by rules as mentioned already in Sec. I. The service management functions are classified into Capability Administration, Capability Configuration, Capability Allocation and Capability Performance Diagnosis, which corresponds to the functionality components presented in Sec. III.

The CapOnt consists of general concepts of capabilities. For a complete capability specification of a specific system, a *System-specific Capability Ontology (SysCapOnt)* must be defined. The SysCapOnt inherits the CapOnt. The service management functionalities presented in Sec. III will need instances of the capability types and the capability parameters part of the SysCapOnt for CA, CC, AL and PD.

B. Representation

OWL [16] and OWL/XDD [18] are used to represent the capability concepts. OWL is a standard Web ontology language, which provides a rich set of constructors. The capability types, the non-rule-based parameters, their relations and instances are expressed by *owl:Class*, *owl:Class*, *owl:ObjectProperty* and *OWL instances* respectively. The values of non-rule-based parameters are specified by *owl:DatatypeProperty* that is called *hasValue*.

However, OWL is limited when it comes to describe the rule-based capability concepts. OWL/XDD, a rule-oriented knowledge representation, is then required to express such concepts. OWL/XDD extends ordinary XML-based elements by incorporation of *variables* for an enhancement of expressive power and representation of implicit information into so called *XML expressions*. The ordinary XML-based elements – XML expressions without variables – are called *ground XML expressions*. The variables are classified into five types as explained in Table I. A variable is prefixed by '\$T:' where T denotes its type.

Table I. The Variable Types

Variable type	Pre fix	Instantiated to
N-variable	\$N:	An element tag name or an attribute name
S-variable	\$S:	An element value or an attribute value
P-variable	\$P:	A sequence of zero or more attribute-value pairs
E-variable	\$E:	A sequence of zero or more XML expressions
I-variable	\$I:	Part of XML expressions

A rule-based parameter and a service management function are represented as an XML clause of the form:

$$H \rightarrow B_1, .., B_m \{C_1, .., C_n\} \qquad (1)$$

where $m, n \geq 0$, H and B_i are XML expressions, and each of C_i is a pre-defined XML condition on the XML clause. H is called the *head* of the clause, while the set of B_i and C_i is the *body* of the clause. When the body is empty, such a clause is referred to as an *XML unit clause* and the symbol '$\rightarrow$' will be omitted. Hence an XML-based element or document can be mapped directly onto a ground XML unit clause.

V. Reasoning Procedure

The reasoning procedure begins with an XML expression based query Q. Then, an XML clause, called query clause, is formulated from the XML expression as follows:

$$Q \rightarrow Q \qquad (2)$$

The XML expression Q represents the constructor of the expected answer which can be derived if all conditions in the body of the clause hold. However, if one or more XML expression bodies still contain the variables, these variables must be matched and resolved from other unit clauses and non-unit clauses.

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

$$R_1: \; H \;\rightarrow\; B_1, B_2, C_1 \qquad (3)$$

If the XML structure of the body Q and the head H of the clause R_1 match without violating the condition C_1, the body Q will be transformed into B_1 and B_2. All 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:

$$Q^* \;\rightarrow\; B_1^*, B_2^* \qquad (4)$$

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

The transformation ends when either 1) the query clause has been transformed into a unit clause or 2) there is no clause R_x that can transform the current bodies of the query clause. If the constructor Q is transformed successfully into Q_f that contains no variable, the reasoning procedure ends and the desired answered is obtained.

VI. Case Study

This section describes an intelligent academic conference room example. A *conference service system* provides video transfer services, and users can download and watch present presentations of accepted papers, present tutorials, and old videos of the presentations and tutorials from previous years. The entire video files will be transferred to the user devices before they can start. In this scenario, the services are classified by the video types: the present presentation, the present tutorial, and the old video. The old video has low priority. The users access the services from their wireless communication devices. However, when the number of users increases, the bandwidth usage becomes too high. When this occurs, the services should be degraded gracefully. This means only high priority services will be available, but they are operating regularly.

The conference service system has adaptability features that deploy and instantiate video transfer service components in capable nodes. Also, it disables the old video transfer services when the wireless bandwidth utilization is greater than a maximum limit, and enables them when the bandwidth utilization is less than a minimum limit.

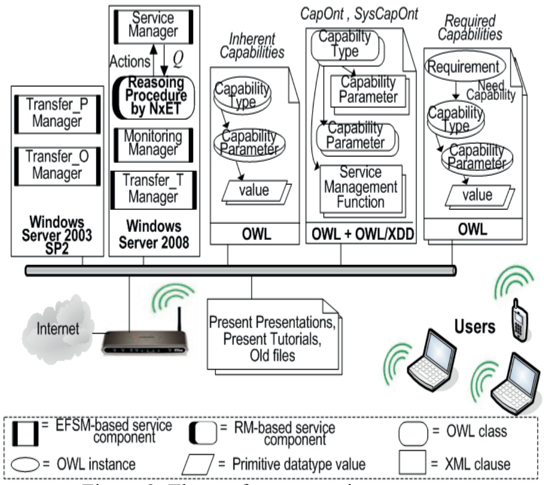


Figure 3. The conference service system.

The conference service system consists of EFSM-based service components and an RM-based service component in nodes as illustrated in Fig. 3. The EFSM-based service components are *Service Manager*, *Monitoring Manager*, *Transfer_P Manager*, *Transfer_T Manager* and *Transfer_O Manager*. A System-specific Capability Ontology (SysCapOnt) will be presented in Sec. VI.A. With reference to the service functionality architecture in Sec. III, the Service Manager implements part of Capability

Administration, Capability Configuration, Deployment and Instantiation, Capability Performance Diagnosis, Capability Allocation and Service Adaptation. The Monitoring Manager implements part of Capability and Service Monitoring, and the RM-based service component is the reasoning procedure used by the Service Manager. The Transfer_P Manager, Transfer_T Manager and Transfer_O Manager offer the transfer services for the following video types respectively: the present presentation, the present tutorial, and the old video.

The Service Manager and Monitoring Manager are instantiated by the administrator. These managers play an important role to attain the adaptability features as mentioned above. The Service Manager has sub functions as follows:

a) Discover available nodes within the conference room.

b) Call the reasoning procedure and get an action: request the Monitoring Manager to monitor a set of capability types and parameters: {$S:CapabilityType, {$S:CapabilityParameter}}, from the nodes every interval.

c) Call the reasoning procedure and get an action: select capable nodes: {$S:Node}, that satisfy the capability requirements of the Transfer_P Manager, Transfer_T Manager and Transfer_O Manager.

d) Deploy and instantiate the Transfer_P Manager, Transfer_T Manager and Transfer_O Manager in the capable nodes.

e) Get the monitored capability instances: {$S:CapabilityTypeInstance, {$S:CapabilityParameterIn-stance, $S:value}}, from the Monitoring Manager.

f) Reallocate the bandwidth every interval by:

- First, call the reasoning procedure and get an action: *{$S:RellocateAction}* to disable or enable the low priority services.
- Second, share the bandwidth equally for every available service.

In b), c) and f), the Service Manager calls the reasoning procedure with an input XML expression based query Q. Using the CapOnt, SysCapOnt, inherent capabilities and require capabilities, the reasoning procedure transforms the query clause (2) to obtain the actions and the instantiations of the variables. Then, these actions and instantiations of the variables are returned to the Service Manager.

The Monitoring Manager gets the monitoring request, queries the parameters' values from the nodes, and updates these values to the Service Manager.

A. System-specific Capabilty Ontology (SysCapOnt)

According to the standardized information models discussed in Sec. II, SNMP MIB and CIM schema have already a rich set of defined capability types and parameters. We choose the SNMP MIB as the basis for the definition of the capability types and the non-rule-based parameters in the SysCapOnt because of the extensive implementation of SNMP agents in different types of devices.

Note that this given ontology as illustrated in Fig. 4 is only for demonstration purpose. The capability types focused are network interface and operating system. These capability types and their parameters are found in Host Resource MIB [19] and MIB II [20].

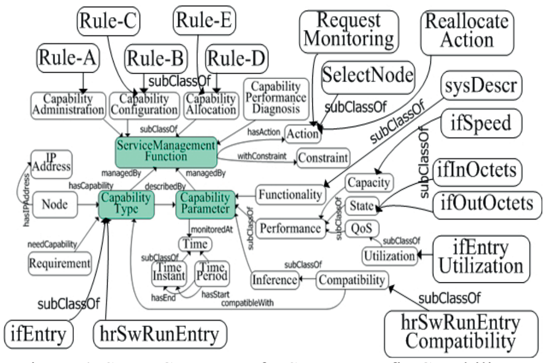


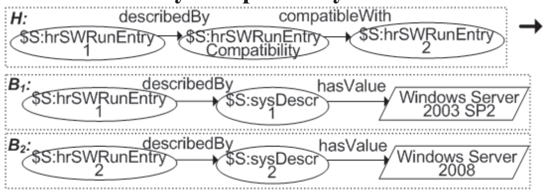
Figure 4. Some Concepts of a System-spefic Capability Ontology.

A network interface is defined by the object type *ifEntry*. It consists of several parameters, such as *ifInOctets*, *ifOutOctets* and *ifSpeed*. *ifInOctets* is the total number of octets received on the interface, whereas *ifOutOctets* refers to the total number of octets transmitted from the interface. *ifSpeed* is the maximum bandwidth of the interface. The parameter *ifEntryUtilization*, a subclass of the Utilization, defines the bandwidth utilization of the interface. An operating system is defined by the object type *hrSWRunEntry*, and its type and version are described by the object type *sysDescr*. The parameter *hrSWRunEntry- Compatibility*, a subclass of the Compatibility, specifies the compatibility between operating systems.

The service management functions are defined by *Rule-A – Rule-E*. These rules, the hrSWRunEntryCompatibility and the ifEntryUtilization are represented in the graphical XML clauses as below. The notations used are:

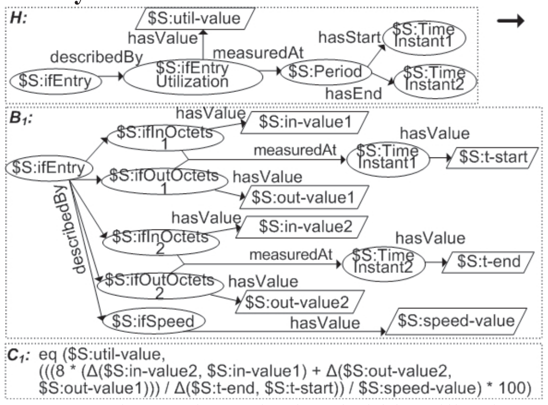
▢ = OWL class ⬭ = OWL instance ▱ = Primitive datatype value

hrSWRunEntryCompatibility xml clause:



It can be read as: Windows Server 2003 SP2 is compatible with Windows Server 2008.

ifEntryUtilization xml clause:

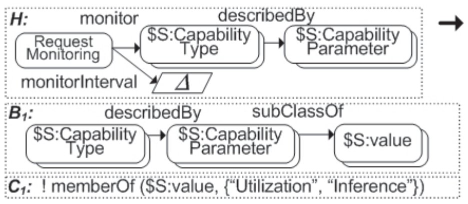


It can be read as: the bandwidth utilization *$S:util-value* during an interval, between *$S:t-start* and *$S:t-end*, depends on other parameters as this expression,

$$\$S\text{:util-value} = (((8 * (\Delta(\$S\text{:in-value2},\$S\text{:in-value1}) + \Delta(\$S\text{:out-value2},\$S\text{:out-value1}))) / \Delta(\$S\text{:t-end},\$S\text{:t-start})) / \$S\text{:speed-value}) * 100 \qquad (5)$$

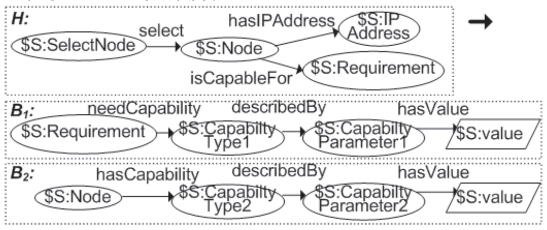
where *Δ($S:in-value2,$S:in-value1)*, *Δ($S:out-value2, $S:out -value1)* are the numbers of bytes received and transmitted via the network interface during an interval, whereas *$S:speed-value* is the maximum bandwidth.

Rule-A xml clause:



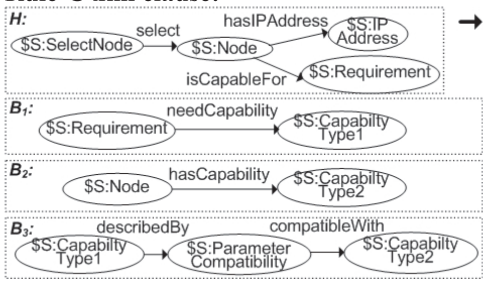
It can be read as: request for the monitoring of *{$S:CapabilityType, {$S:Capability-Parameter}}* every interval (Δ), if *$S:CapabilityParameter* is not the rule-based parameters: the Utilization- and the Inference parameter.

Rule-B xml clause:



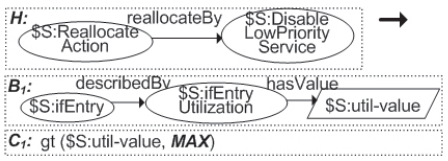
It can be read as: select a node *$:Node* as a capable node for the requirement *$S:Requirement* if it has capability types, parameters and values as same as required.

Rule-C xml clause:



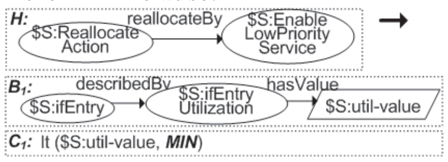
It can be read as: select a node *$:Node* as a capable node for the requirement *$S:Requirement* if it has capability types as compatible as required.

Rule-D xml clause:



It can be read as: reallocate the bandwidth by disabling the low priority services, if the bandwidth utilization is greater than a maximum limit (MAX).

Rule-E xml clause:



It can be read as: reallocate the bandwidth by enabling the low priority services, if the bandwidth utilization is less than a minimum limit (MIN).

B. Simulation Results

We simulated our scenario by assuming that the nodes have the SNMP agents executing. The maximum wireless bandwidth is 54Mbps. The number of users is 100. Every user generates 2-5 service requests to transfer the videos *randomly*. All video files are the same size; 300Mb, and are transferred with the same maximum rate; 1Mbps. The interval (Δ) to monitor capabilities as well as to reallocate the bandwidth is 30 sec.

Our simulation is set for two cases (I, II). Both cases set low priority for the old videos as already mentioned. In case I, the system executes without the Rule-D and the Rule-E. The requests will wait if there is no bandwidth left. When there is released bandwidth, it is shared and given to all requests; processing requests and waiting requests, of the high priority videos before the low priority videos. In case II, the system executes with the Rule-D and the Rule-E. The old video transfer services will be disabled and enabled according to the bandwidth utilization. The maximum limit (MAX) and the minimum limit (MIN) are set as 80% and 60%. The average transfer times (Avr.T) in both cases are presented in Table II.

Table II. The average transfer times

	Avr.T of The Present Presentations	**Avr.T of The Present Tutorials**	**Avr.T of The Old Videos**
Case I	6.82 min	6.26 min	9.73 min
Case II	5.76 min	5.39 min	14.95 min

In Case II, the system degrades the services more efficiently than in Case I. The system can offer an adequate transfer of the high priority videos. The average transfer times are 5.76 min and 5.39 min, which are faster than in Case I ≈15.5% and ≈13.9% respectively. These %values indeed vary according to the arrival distribution, Δ, MAX and MIN. However, the average transfer time of the low priority videos in Case II (14.95 min) is greater than in Case I (9.73 min). This is because in Case II the reasoning procedure executes the rules and suggests the action to pause their transfers when the bandwidth utilization > 80%. The bandwidth that has been used is released, and is given for transferring the high priority videos. But in Case I the bandwidth is released only when the transfers finish.

VII. Conclusions

Capability Ontology (CapOnt) comprising the concept capability types, capability parameters and service management functions is proposed. The rule-based parameters values, e.g. the Utilization and the Compatibility, are generated dynamically by the presented reasoning mechanism. The capability types and the non-rule-based capability parameters can be defined based on the existing standardized information models. An intelligent conference room example, where the Capability Ontology combined with the reasoning mechanism has proved useful, is presented. A System-specific Capability Ontology (SysCapOnt) for a conference service system is also given. The conference service system comprises the adaptation in case the bandwidth utilization is greater than or lower than the limits. Using the proposed rule-based service management functions, the system (re)-allocates the bandwidth to the high priority service classes more efficiently.

References

[1] F. A. Aagesen, P. Supadulchai, C. Anutariya, and M. M. Shiaa, "Configuration Management for an Adaptable Service System," In Proc of IFIP Int. conf. on Metropolitan Area Networks, Architecture, Protocols, Control and Management, Vietnam, April, 2005.

[2] K. McCloghrie, D. Perkins, and J. Schoenwaelder, "Structure of Management Information Version 2," RFC 2578, 1999. Available at: http://www.ietf.org/rfc/rfc2578.txt [Last accessed June 2010].

[3] DMTF, "Common Information Model (CIM) Standards," Available at: http://www.dmtf.org/standards/cim/ [Last accessed June 2010].

[4] R. Studer, V. R. Benjamins, and D. Fensel, "Knowledge Engineering: princicples and methods," in Data & Knowledge Engineering, vol. 25, pp. 161-197, 1998.

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

[6] W. Stallings, SNMP, SNMPv2, SNMPv3, RMON 1 and 2. The 3rd edition, Addison-Wesley, 1999.

[7] C. Elliott, D. Harrington, J. Jason, J. Schoenwaelder, F. Strauss, and W. Weiss, "SMIng Objectives," RFC 3216, 2001.

[8] ITU-T Recommendation X.722, Information technology - Open Systems Interconnection, "Structure of management information: Guidelines for the definition of managed objects," January 1992.

[9] J. E. López de Vergara, V. A. Villagrá, and J. Berrocal, "On the Formalization of the Common Information Model Metaschema," In Proc of the 16th IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM). LNCS, vol. 3775/2005.

[10] J. E. López de Vergara, V. A. Villagrá, and J. Berrocal, "Applying the web ontology language to management information definitions," In IEEE Commun. Mag., vol. 42, no. 7, pp.68–74, Jul. 2004.

[11] J. Keeney, D. Lewis, D. O'Sullivan, A. Roelens, A. Boran, and R. Richardson, "Runtime semantic interoperability for gathering ontology-based network context," In Proc of IEEE/IFIP Network Operations and Management Symposium (NOMS'06), Canada, 2006.

[12] A. K. Y. Wong, P. Ray, N. Parameswaran, and J. Strassner, "Ontology mapping for the interoperability problem in network management," In IEEE Journal Sel. Areas Commun. 23(10), pp.2058-2068, 2005.

[13] A. Guerrero, V. A. Villagrá and J. E. López de Vergara, "Ontology-Based Integration of Management Behaviour and Information Definitions Using SWRL and OWL," In Proc of the 16th IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM'05), Spain, 2005.

[14] H. Xu and D. Xiao, "A Common Ontology-Based Intelligent Configuration Management Model for IP Network Devices," In Proc of the 1st Int. Conf. on Innovative Computing, Information and Control (ICICIC'06), China, 2006.
[15] I. Diaz, C. Popi, O. Festor, J. Tourino, and R. Doallo, "Ontological Configuration Management for Wireless Mesh Routers," In Proc of the 9th IEEE Int. Workshop on IP Operations and Management (IPOM'09), Italy, 2009.
[16] W3C, "OWL Web Ontology Language Overview," 2004 Available at: http://www.w3.org/TR/owl-features/ [Last accessed June 2010].
[17] P. Supadulchai and F. A. Aagesen, "Policy-based Adaptable Service Systems Architecture," In Proc of the 21st IEEE Int. Conf. on Advanced Information Networking and Applications (AINA'07), Canada, 2007.
[18] V. Wuwonse and M. Yoshikawa, "Towards a language for metadata schemas for interoperability," In Proc of the 4th Int. Conf. on Dublin Core and Metadata Applications, China, 2004.
[19] S. Waldbusser and P. Grillo, "Host Resource MIB," RFC 2790, 2000, Available at: http://www.ietf.org/rfc/rfc2790.txt [Last accessed June 2010].
[20] K. McCloghrie and M. Rose, "MIB II," RFC 1213, 1991, Available at: http://www.ietf.org/rfc/rfc1213.txt [Last accessed June 2010].

Paper C: OWL-based Node Capability Parameter Configuration

Patcharee Thongtra, Finn Arve Aagesen and Kornchnok Dittawit

Published in
Proceedings of the 18th EUNICE/IFIP WG 6.2, 6.6 International Conference on Information and Communications Technologies

Budapest, Hungary, August 2012

Lecture Notes in Computer Science (LNCS) 7479, pp. 124-135, 2012.

OWL-based Node Capability Parameter Configuration

Patcharee Thongtra and Finn Arve Aagesen

Department of Telematics
Norwegian University of Science and Technology
N-7491 Trondheim, Norway
{patt, finnarve}@item.ntnu.no

Abstract. Node capability parameter configuration is the validation and settings of node capability parameter values according to node capability parameter configuration specification (CapSpc). A node capability is a property of a node required as basis for service implementation. This paper presents a Node Capability Parameter Configuration System (CapCon). Node Capability Ontology (CapOnt) is the basis for CapCon. This paper has focus on CapCon in network management. CapSpc specifies required types, parameters, and parameter values for the node capabilities. OWL and OWL/XDD are used to represent the ontology concepts. The NETCONF framework is applied for the network management functionality. A prototype implementation and a case study including experimental results are presented.

Keywords: Network Management, NETCONF, Capability Configuration.

1. Introduction

Network management has been a topic of study for some decades. Some important network management frameworks are SNMP (Simple Network Management Protocol) [1], WBEM (Web-Based Enterprise Management) [2], WS-Management (Web Services Management) [3] and NETCONF (Network Configuration Protocol) [4]. In this paper, we apply the SNMP concept MIB as a generic concept for the system of managed objects.

A *node capability* is generally defined as a property of a *node* applied as basis for service implementation. *Node capability parameter configuration* is the validation and settings of node capability parameter values according to a *node capability parameter configuration specification* (CapSpc). CapSpc specifies required types, parameters, and parameter values for the node capabilities of all nodes in a domain. Existing network management frameworks as mentioned above do not provide functionality for automatic node capability parameter configuration.

This paper presents a *Node Capability Parameter Configuration System (CapCon).* CapCon identifies which nodes are capable/incapable to meet, or have already met, the requirements of CapSpc. CapCon can set the node capability parameter values on the nodes that can meet the requirements, while generating a report containing node capability types installation instruction for the nodes incapable of satisfying the requirements.

Ontology is a formal and explicit specification of shared concepts [5-6]. Node Capability Ontology (CapOnt) enables the definitions of non-rule-based and rule-based concepts. CapOnt is the basis for the definition of CapSpc. Some MIB concepts will be a subset of CapOnt, but represented differently. There is accordingly a need to transform between CapOnt and MIB. CapCon can be integrated with various network management frameworks. In this paper NETCONF protocol with NETCONF agents are applied. NETCONF was designed for managing configuration data. NETCONF, however, does not provide functionality to efficiently manage configuration data for a set of nodes.

The rest of the paper is organized as follows. Section 2 presents an overview of NETCONF. Functionalities and system architecture of CapCon are described in Section 3. Section 4 presents CapOnt, comprising the concepts, the representation and storage of the ontology concepts, as well as the transformation between the CapOnt concepts and YANG-based concepts. Section 5 presents a prototype implementation of CapCon. The CapOnt node capabilities and the NETCONF MIB considered are based on the SNMP IF-MIB [7]. NETCONF agent is based on the YUMA toolkit [8]. The functionality of YUMA agent was extended to support IF-MIB YANG module [9] and also to enable registration and de-registration of nodes. Section 6 presents related work and Section 7 gives summary and conclusions.

2. NETCONF Management Framework

NETCONF management framework follows the manager/agent model. Managed objects, however, are divided into configuration data and state data. *Configuration data* is writable and is required for transforming a device from its initial default state into the desired state. *State data* is read-only status information and statistics. In the following short descriptions of NETCONF capability, NETCONF configuration datastore, NETCONF protocol, and YANG are given.

NETCONF capability is a set of functionalities that are implemented by both manager and agent. The NETCONF capabilities are declared in <hello> messages sent by each peer during the session initialization. The base NETCONF capability is the minimum set of functionalities each peer must implement. In addition to the base NETCONF capability, there is a set of standardized NETCONF capabilities which the NETCONF agent may support, e.g., :candidate and :notification. The :candidate NETCONF capability indicates that the agent supports the candidate configuration datastore (see below). The :notification NETCONF capability indicates that the agent supports the basic notification delivery mechanisms.

NETCONF configuration datastore is a complete set of configuration data. NETCONF has *running*, *startup,* and *candidate* configuration datastore. *Running* configuration datastore represents the currently active configuration, while *startup* configuration datastore is the configuration that will be used during the next startup. *Candidate* configuration datastore represents a configuration that may be prepared and later committed to become the new running configuration datastore. Only the running configuration datastore is present in the *base* NETCONF capability. The other configuration datastores can be defined by additional NETCONF capabilities.

NETCONF protocol provides a simple RPC mechanism to install, manipulate, and delete configuration data. It has four layers: *content*, *operation*, *RPC,* and *transport*. *Content* layer represents the managed objects. *Operation* layer defines operations that can be invoked as RPC methods such as *get* operation to retrieve the managed object instances and *edit-config* operation to modify a configuration datastore. *RPC* layer provides a transport-independent framing mechanism. Several transport protocols can be used, e.g. SOAP, SSL, and BEEP.

NETCONF MIB concepts are represented by YANG [10], but are still partly based on SNMP MIB. YANG-based MIB is structured into modules. Objects in each module are modeled as YANG nodes in a hierarchical tree. YANG nodes are of four types: *container*, *list*, *leaf* and *leaf-list*. A *container* node as well as a *list* node is an internal YANG node that has no value apart from a set of child nodes. A *leaf* node as well as a *leaf-list* node has a value but no child nodes. A container node and a leaf node have at most one instance. A list node and a leaf-list node may have multiple instances. YANG modules supported by a NETCONF agent are indicated via the <hello> messages. An MIB compiler called libsmi [11] has been implemented to translate SMIv2 MIB modules to YANG modules. As an example of a SNMP-based *NETCONF MIB*, a part of *IF-MIB YANG module (NETCONF IF-MIB)* translated from SNMP IF-MIB [7] is presented in Fig. 1.

```
module IF-MIB {namespace "urn:ietf:params:xml:ns:yang:smiv2:IF-MIB"; prefix
"if-mib";
container interfaces {
leaf ifNumber {type int32; config false; .. }
list ifEntry {key "ifIndex"; leaf ifIndex {type if-mib:InterfaceIndex; .. }
leaf ifType {type ianaiftype-mib:IANAifType; config false; .. }
leaf ifSpeed {type yang:gauge32; config false; .. }
leaf ifAdminStatus { type enumeration {enum up {value 1;} enum down {value 2;}
enum testing {value 3;}} config true; .. } .. }}} .. }
```

Fig. 1. A part of IF-MIB YANG module.

3. Functionalities and System Architecture

CapCon architecture as illustrated in Fig. 2 has three functionality components, two repositories and NETCONF agents. The functionality components are Node Capability Administrator *(NCA),* Node Capability Monitor *(NCM*) and Node Capability Parameter Modifier *(NCD). NCA* is concerned with the registration and de-registration of nodes and their node capabilities. *NCM* monitors node liveness and node capability- types, parameters, and parameter values. *NCD* is responsible for the validation and settings of node capability parameter values according to CapSpc. The repositories are *Node Capability Type Repository* (NCRep) and *Inherent Node Capability Repository* (INRep). NCRep stores the ontology concepts. INRep stores the existence and liveness of nodes as well as inherent node capabilities. *Inherent node capabilities* are observed node capability *instances*. The NETCONF agent registers and de-registers with NCA using <register> and <deregister> messages. Both messages contain the node IP address.

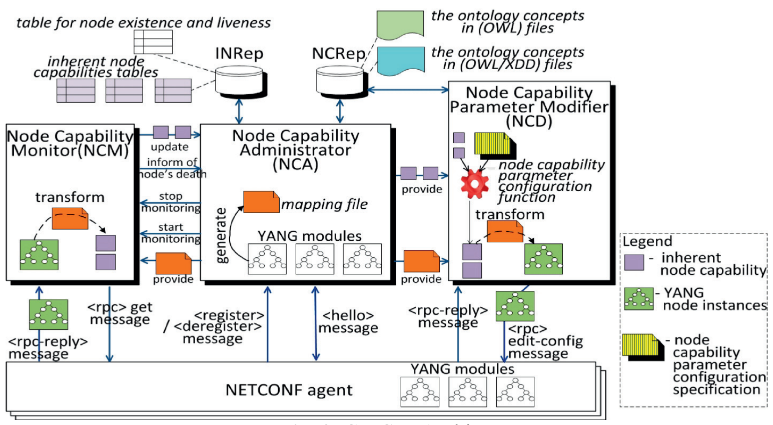


table for node existence and liveness
inherent node capabilities tables
INRep
NCRep
the ontology concepts in (OWL) files
the ontology concepts in (OWL/XDD) files
Node Capability Parameter Modifier (NCD)
Node Capability Monitor(NCM)
Node Capability Administrator (NCA)
update
inform of node's death
stop monitoring
start monitoring
provide
transform
mapping file
generate
YANG modules
node capability parameter configuration function
Legend
- inherent node capability
- YANG node instances
- node capability parameter configuration specification
<rpc> get message
<rpc-reply> message
<register> / <deregister> message
<hello> message
<rpc> edit-config message
NETCONF agent

Fig. 2. CapCon Architecture.

NCA creates INRep. First, NCA creates tables for storing the inherent node capabilities and a table for storing node existence and liveness in INRep. Then, if NCA gets a <register> message from a NETCONF agent, it will record the node's existence. If NCA gets a <hello> message from a NETCONF agent, it will parse the message and identify which YANG modules are supported. NCA will load these YANG modules into its memory and create a mapping file for the transformation between CapOnt and MIB. This mapping file will be sent to NCM and NCD. Also, NCA will send a monitoring request to NCM. If NCA gets a <deregister> message from a NETCONF agent, it will remove the node existence record and send a request to NCM to stop monitoring the node liveness and the inherent node capabilities. In addition, NCA provides the current view of inherent node capabilities in INRep to the other functionality components.

NCM gets monitoring requests from NCA and regularly sends an <rpc> message containing the NETCONF get operation to the agent. Then, if NCM gets an <rpc-reply> message with YANG node instances, it will inform NCA that the node is alive. Also, NCM will use the mapping file to transform the YANG node instances to the inherent node capabilities, and update NCA of such inherent node capabilities. If NCM does not get a <rpc-reply> message for a certain period, it will inform NCA that the node is dead.

NCD regularly identifies which registered nodes are capable/incapable of meeting, or have already met, the requirements of node capability types, parameters and parameter values. A node having the required types, parameters and parameter values have already met the requirements; a node having only the required types and parameters are capable of meeting the requirements. On the other hand, a node that lacks the required types and parameters are incapable of meeting the requirements. NCD sets the node capability parameter values for those nodes that can meet the requirements, while generating a report for further installation of the required node capability types for those previously incapable of satisfying the requirements. NCD operations are based on Equivalent Transformation (ET) [12]. CapSpc, CapOnt and the inherent node capabilities are inputs. Outputs are one or more of nodes' IP address and the NETCONF edit-config operations with node capability parameter instances and required values. NCD will use the mapping file to transform the node capability parameter instances to YANG node instances, and send the <rpc> messages to the corresponding agents. Each of these messages contains the NETCONF edit-config operation with the YANG node instance.

4. Node Capability Ontology

This section defines CapOnt concept structure as well as the language representation and storage of these concepts. The projection of MIB concepts into CapOnt as well as the transformation between CapOnt and MIB concepts is also considered.

4.1 Upper Node Capability Ontology

The generic structure of CapOnt is illustrated in Fig. 3. This generic concept structure is denoted as *Upper Node Capability Ontology*. *Inference* parameters define logical relations to other node capability types. Parameters can be defined by *rules*. The relations between the node, node capability types and node capability parameters are referred to as: *ownership relation* between the node and node capability type, *constitution relation* between two node capability types; and *description relation* between the node capability- type and parameter. A *service management function* is an action with constraints and is defined by a rule. For a specific system, specific CapOnt concepts are defined.

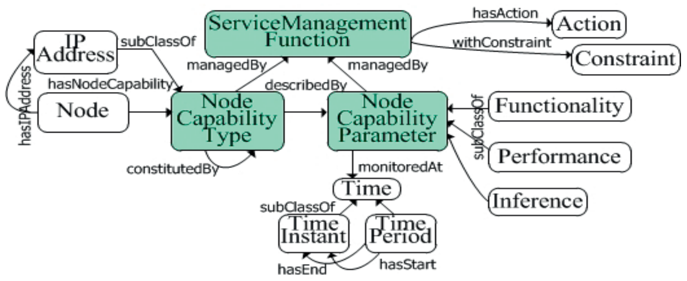


Fig. 3. Upper Node Capability Ontology.

4.2 MIB Concepts in CapOnt

NETCONF MIB concepts are represented by YANG, but are partly based on SNMP MIB. SNMP MIB objects will in CapOnt be represented as node capability types and non-rule-based parameters. SNMP MIBs are structured in a logical tree. SNMP tables are defined by managed objects on 3 MIB node levels. The top level node is here denoted as a *table root node*, the second level node as a *table access node*, and the lowest level nodes as *columnar leaf nodes*.

To create CapOnt from a SNMP MIB group, we consider three cases: 1) The inner node X is neither a table root node nor a table access node, 2) The leaf node Y is not a columnar leaf node, and 3) The inner node Z is a table root node with a table access node A and columnar leaf nodes {Ci}. For Case 1) the pair of connecting inner nodes Xi and Xj is projected to two node capability types (Xi and Xj) with a constitution relation from Xi to Xj. In Case 2) the pair of inner node X and leaf node Y is projected to a node capability type X, a non-rule-based parameter Y, and a description relation from X to Y. In Case 3) the pair of inner node X and table root node Z with a table access node A and columnar leaf nodes {Ci} is projected to two node capability types (X and A), non-rule-based parameters {Ci}, and there is a constitution relation from X to A, and description relations from A to each Ci.

4.3 Representation and Storage of CapOnt

Non-rule-based concepts are expressed by OWL [13]. OWL is a standard Web ontology language, which provides a rich set of constructors to define concepts. However, OWL is limited when it comes to describe rule-based concepts. So, *rule-based concepts* are expressed by OWL/XDD [14]. OWL/XDD extends ordinary XML-based elements by incorporation of variables and rule-based concepts. A rule-based concept in CapOnt is expressed by an OWL/XDD XML clause of the form:

$$H \rightarrow B_1, .., B_m, \{C_1, .., C_n\} \qquad \textbf{(1)}$$

where $m, n \geq 0$, H and B_i are XML expressions, and each of C_i is a pre-defined XML condition on the clause. H is called the *head* of the clause while the set of B_i and C_i is the *body* of the clause. An XML expression is an XML-based element or document embedding zero or more variables. The upper ontology concepts as well as the CapOnt concepts for the prototype framework described in Sec.5 are given at http://tapas.item.ntnu.no/wiki/index.php/CapOnt.

CapSpc is based on CapOnt concepts. CapSpc is expressed by one or more OWL/XDD XML clauses, where XML expressions of these clauses are instances of the CapOnt concepts with variables.

The ontology concepts are stored in NCRep. The inherent node capabilities are *instances* of the *CapOnt non-rule-based concepts*. They are stored in a relational database, and are inputs for dynamically setting variables in the CapOnt rule-based concepts with suitable values. NCA is responsible for creating tables in the database. These tables denoted as *inherent node capabilities tables* are generated from the *representation* of the *CapOnt non-rule-based concepts*, using the *rules i)-iv)* below.

i). An *OWL Class class_x* will be mapped to a *Table class_x*, if it has either one of the properties: hasNodeCapability, constitutedBy, or describedBy. The Table class_x is assigned an auto-numbered Primary key named ID and a TIMESTAMP Column named *MonitoredAt* for recording the row-creation/update time. The Table "Node" is also assigned a Column IPAddress.
ii). For Class(*class_x* .* restriction(*property_i* allValuesFrom(unionOf(*set_of_classes*))) .*); *class_y* ∈set_of_classes, *Foreign key class_x_ID* will be generated in *Table class_y* referring to the Primary key in *Table class_x*, if property_i ≡ hasNodeCapability || constitutedBy.
iii). For Class(*class_x* .* restriction(*property_i* allValuesFrom(unionOf(*set_of_classes*))) .*); *class_y* ∈set_of_classes, *Column class_y* will be generated in *Table class_x*, if property_i ≡ describedBy.
iv). For Class(*class_y* .* restriction(*property_i* allValuesFrom(*primitive_datatype*)) .*), *primitive_datatype* will be converted to an SQL primitive datatype and become the *Column class_y* datatype, if property_i ≡ hasValue.

The expression (2), used in the rule ii) – iv), expresses a restriction of an OWL Class *class_x* on *property_i*, in which the property's range is restricted to *property_range_j*.

Class(*class_x* .* restriction(*property_i* allValuesFrom(*property_range_j*)) .*); **(2)**
property_range_j ≡ unionOf(*set_of_ classes*) || *primitive_datatype*

The node *instances* and the node capability type *instances* are stored in the *Tables* generated from the rule i). The non-rule-based parameter *instances* are stored in the *Columns* generated from the rule iii).

4.4 Transformation between CapOnt and NETCONF MIB

NCA is responsible for creating a *mapping file* to be used for the transformation between *CapOnt* and the corresponding YANG nodes in NETCONF MIB. A capability type, which is constituted by at least another node capability type, is mapped to a YANG *container* node. A capability type, which is not constituted by others, is mapped to a YANG *container* node or a YANG *list* node. A non-rule-based parameter is mapped to a YANG *leaf* node.

A YANG module defines a tree of YANG nodes. A *path* defines the YANG node's position relative to the root. NCA assigns a path to all YANG nodes, defined as a sequence of YANG nodes' types and names, separated by slashes, starting from the root to the YANG node assigned the path. NCA finds the corresponding YANG node for each of the node capability types and non-rule-based parameters by matching the ontology concept's name and the YANG node's name. When SNMP MIB managed objects are the basis of CapOnt concepts, there is a one-to-one mapping between these concepts and the YANG nodes. Otherwise, other logics for matching are required. A set of pairs of an OWL Class, expressing a node capability type or a non-rule-based parameter, and a YANG node's path is then generated and stored in the mapping file. With respect to the IF-MIB YANG module example in Fig. 1, an example pair for the parameter *ifAdminStatus* and the corresponding leaf node *ifAdminStatus*, locating under the list node *ifEntry* and the container node *interfaces* is {<Class rdf:ID="ifAdminStatus">, container interfaces/list ifEntry/leaf ifAdminStatus}.

5. A Prototype Node Capability Parameter Configuration System

5.1 General

The system consists of a load balancer and a cluster of web servers as illustrated in Fig. 4. The load balancer captures the HTTP-based user requests and forwards them to connected web servers on a round robin basis. The number of connected web servers during an interval is dynamic and depends on the total number of user requests during the previous interval. CapCon connects and disconnects web servers to/from the load balancer. Web servers will go into hibernation for energy conservation in case of an inactive time period, i.e., no incoming user requests forwarded from the load balancer or user requests have already been responded.

A prototype is implemented using JAVA programming language. NCA, NCM, and NCD are implemented based on TAPAS platform [15], which enables NCA, NCM, and

NCD to be instantiated and executed in different nodes. NCRep is a physical directory, while INRep is realized by a PostgreSQL database. The NETCONF MIB considered is NETCONF IF-MIB [9]. NETCONF agent is based on YUMA toolkit [8]. YUMA agent functionalities have been extended to support NETCONF IF-MIB and also to provide registration and de-registration of nodes as explained in Section 3.

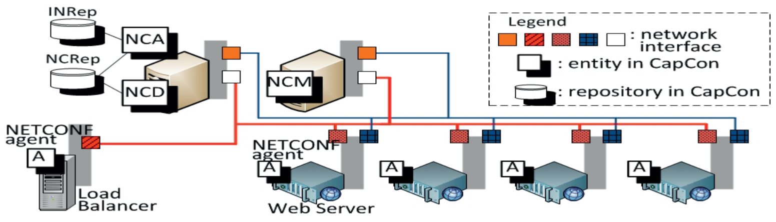


Fig. 4. A web-based application example.

5.2 Node Capability Ontology Concepts

The node capability types and non-rule-based parameters in CapOnt are projected from the *interfaces MIB group* in *SNMP IF-MIB* [7]. In this MIB group, there is a managed object *ifNumber* and a table *ifTable*. *ifNumber* specifies the number of network interfaces. For *ifTable*, *ifEntry* is the conceptual row representing a particular network interface. There are 22 columnar objects of which 18 have Status = current. As examples *ifIndex* has values unique for each network interface, *ifInOctets* gives the total number of octets received on the network interface, and *ifSpeed* defines the maximum bandwidth of the interface in bit/second. The object *ifAdminStatus* defines the state of the interface which can either be ‘up(1)’, ‘down(2)’, or ‘testing(3)’.

From the interfaces MIB group, CapOnt has two node capability types and 19 non-rule-based parameters. The node capability types are *interfaces* and *ifEntry*. The non-rule-based parameters are *ifNumber* and all of those 18 mentioned columnar managed objects. These concepts’ instances are stored in the inherent node capabilities tables generated by using the rules in Section 4.3. CapOnt has also a rule-based parameter *ifInUtilization* and two service management functions. *ifInUtilization* is the inbound bandwidth utilization of a network interface in percentage. The service management functions’ actions are setting a network interface’s state. In this paper, the actions are specialized as the NETCONF edit-config operation to set the parameter *ifAdminStatus* value.

The CapOnt non-rule-based concepts in an OWL file as well as the rule-based ifInUtilization parameter and service management functions in OWL/XDD XML clauses are presented at http://tapas.item.ntnu.no/wiki/index.php/CapOnt.

5.3 Node Capability Configuration Specification Concepts

CapSpc for the CapCon prototype is expressed in Fig. 5. It is expresses that “A node with a 100Mbps network interface acting as the load balancer is required. Four nodes with two network interfaces acting as the web servers are required. The first network interface of the web server that *ifIndex* = 1 is required for the connection with CapCon. This network interface's state is always ‘up’. The second network interface of the web server that *ifIndex* = 2 is required for the connection with the load balancer, and its state will be changed dynamically.

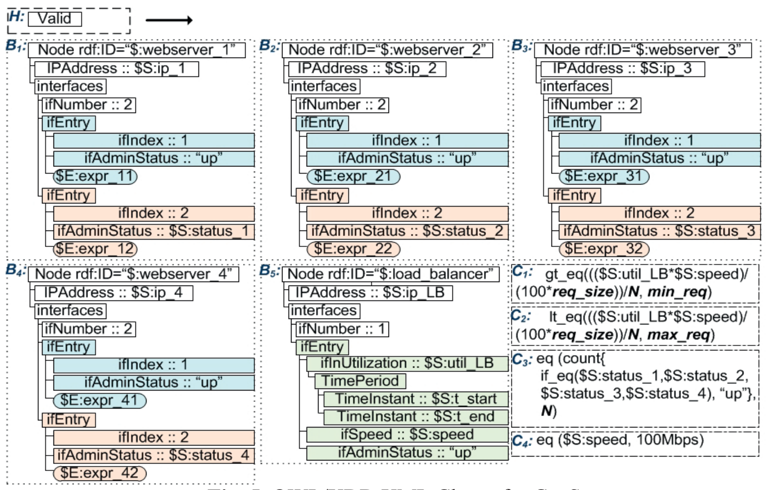


Fig. 5. OWL/XDD XML Clause for CapSpc.

$S:util_LB is the value of the ifInUtilization of the load balancer during the previous interval, between *$S:t_start* and *$S:t_end* second. The total number of user requests during the previous interval can be calculated from: ($S:util_LB*$S:speed)/(100*req_size), where *req_size* is the user request size in bit, and *$S:*speed is the maximum bandwidth of the load balancer (100Mbps). *N* is the appropriate number of web servers connecting with the load balancer during an interval. This means the number of user requests per web server must be between *min_req* and *max_req*, where *min_req* is the estimated *minimum concurrent user requests* per second that a web server should process, and *max_req* is the *maximum concurrent user requests* per second that a web server can process."

5.4 Experimental Results

Two sets of experiments (I, II) have been conducted to illustrate the result of automatic parameter values setting. In this case study, the parameter considered is ifAdminStatus. The evaluation is based on the level of energy conservation achieved from the disconnection of web servers. In both experiments, there is a node with a 100Mbps network interface being the load balancer, four nodes with two network interfaces being the web servers. A *period of inactivity* before hibernate mode of the web servers becomes activated is two minutes. A *cost unit* determines the amount of energy usage: 1 cost unit during normal mode and 0.3 cost units during hibernation mode. The time interval for the polling of NCM as well as the identification and value settings in NCD is 30 seconds. The controlled variables in the CapSpc in Section 5.3 are set as req_size = 500 bytes, min_req = 500 req/sec, and max_req = 1000 req/sec. In these experiments inbound traffic of the load balancer is controlled, and is generated only from the user requests.

In experiment I, the user requests were randomly generated at the rate between 1000-4000 req/sec. In experiment II, the user requests were generated according to the time of day: 2001-4000 req/sec during 08:00-22:00 and 1000-2000 req/sec during 22:00-08:00. During the execution, the web servers were capable to meet, or had already met, the requirement. Based on the CapSpc, the CapOnt concepts, and the inherent node capabilities from NCA, the node capability parameter configuration function in NCD calculated the ifInUtilization value of the load balancer, and validated if the number of connecting web servers was the same as required. When it was not, this function returned one or more of randomly selected web servers' IP address and the NETCONF edit-config operations to NCD. Each of the operations contains the parameter ifAdminStatus instance and required value (‘up’/‘down’). NCD sent the <rpc> messages to the corresponding agents, and consequently such web servers were re-connected/disconnected.

The experiments have been carried out with the same total number of user requests for both experiments. The results from both experiments show that three out of four web servers were disconnected and entered hibernating mode. Table 1 shows the number of times each web server was disconnected and hibernated. It also shows that the time percentage that a web server in experiment II enters hibernation after getting disconnected is more than its counterpart in experiment I, since the user requests rate in experiment II is more certain. We can indicate that the usage of CapCon for the “time-based” user request rate results in higher energy conservation.

Table 1. Experimental results.

	Experiment I	Experiment II
Server_1: Disconneted / Hibernated	258/32	276/122
Server_2: Disconneted / Hibernated	39/11	20/19
Server_3: Disconneted / Hibernated	13/8	5/5
Conserved energy (% of saving cost unit per day)	16.09%	41.58%

6. Related Work

Two aspects regarding system architecture and information model are considered in this paper. Most of the recent works related to NETCONF-based configuration management systems, however, focus only on one of these aspects. Some examples are found in [16-19]. The work in [16-17] focuses on the system architecture aspect, [18] mainly considers the NETCONF MIB represented by YANG, and both aspects are discussed in [19]. Cui et al. [16] present a detailed design of NETCONF manager and agent. A proxy configuration file is used to specify the agents in the entire managed devices. In CapCon, the NETCONF agents are able to register nodes and the node capabilities without the need of configuration file. Liu et al. [17] improve traditional network management system by adding a simple judging function to dynamically decide whether the NETCONF protocol is supported on the managed devices. If the NETCONF protocol is supported, NETCONF operations are used. Otherwise, SNMP operations are used. Nataf and Festor [18] integrate the NETCONF MIB represented by YANG into the NETCONF agent from the ENSUITE open source framework, and implement a browser to retrieve and edit the configuration data. Elbadawi and Yu [19] present the design and implementation of a configuration validation system using Erlang programming language. Comparing with CapCon, the system provides the validation without the parameter value settings.

7. Conclusions

Node Capability Parameter Configuration System (CapCon) which enables automatic node capability parameter configuration is presented. Two aspects including centralized architecture and Node Capability Ontology (CapOnt) are discussed in details. The CapOnt concept representations and the transformation between CapOnt and SNMP-based NETCONF MIB are presented. In the case study, the prototype web-based application integrated with CapCon has been implemented. The automatic parameter value setting by CapCon dynamically adapts the number of active web servers.

CapCon provides flexibility in terms of the applied frameworks. The CapOnt concepts as well as the CapSpc concepts can be added, modified and removed during the system runtime. CapCon also enables automatic discovery of nodes and their node capabilities. In this paper, CapCon is integrated with NETCONF. The flexible nature of the system enables integration with various network management frameworks. However, the future work can be focused on developing a decentralized architecture of CapCon.

References

1. Subramanian, M.: Network Management - Principles and Practice. Addison-Wesley, 2000.
2. DMTF, Web-Based Enterprise Management, http://dmtf.org/standards/wbem
3. DMTF, Web Services Management, http://dmtf.org/standards/wsman
4. Enns, R.: NETCONF Configuration Protocol, IETF RFC 4741, Dec 2006.
5. Studer, R., Benjamins, V. R., Fensel, D.: Knowledge Engineering: principles and methods. In: Data & Knowledge Engineering, vol. 25, pp. 161-197, 1998.
6. Thongtra, P., Aagesen, F. A.: Capability Ontology in Adaptable Service System Framework. In: Proc. of 5th Int. Multi-Conference on Computing in the Global Information Technology, Spain, Sep 2010.
7. McCloghrie, K., Kastenholz, F.: The Interfaces Group MIB, IETF RFC 2863, Jun 2000.
8. YUMA - YANG-based Unified Modular Automation toolkit for the NETCONF protocol, http://www.NETCONFcentral.org/yuma
9. IF-MIB YANG module, http://www.netconfcentral.org/modulereport/IF-MIB
10. Bjorklund, M.: YANG - A Data Modelling Language for the Network Configuration Protocol (NETCONF), IETF RFC 6020, Oct 2010.
11. libsmi - A Library to Access SMI MIB Information, http://svn.ibr.cs.tu-bs.de/projects/libsmi
12. Akama, K., Shimitsu, T., Miyamoto, E.: Solving Problems by Equivalent Transformation of Declarative Programs. In: Journal of the Japanese Society of Artificial Intelligence, vol. 13, pp. 944-952, 1998.
13. W3C, OWL Web Ontology Language Overview, 2004, http://www.w3.org/TR/owl-features/
14. Wuwonse, V., Yoshikawa, M.: Towards a language for metadata schemas for interoperability. In: Proc. of 4th Int. Conf. on Dublin Core and Metadata Applications, China, 2004.
15. TAPAS Platform - A support system for deployment, execution and management of service systems defined by the TAPAS architecture concepts, http://tapas.item.ntnu.no/wiki/index.php/TAPAS_Platform
16. Cui, J., Jia, K., Wu, L., Chen, C., Lai, M.: The Design of the Network Configuration Management based on NETCONF protocol. In: Proc. of Int. Conference on Applied Informatics and Communication (ICAIC 2011), Xi'an, China, Aug 2011.

17. Liu, L., Xiao, D., Dong, B., Shen, Q.: Implementation of the management of SNMP/NETCONF network devices for the next generation NMS. In: Proc. of 2nd Int. Conference on Electrical and Control Engineering (ICECE 2011), Yichang, China, Sep 2011.
18. Nataf, E., Festor, O.: End-to-end YANG-based Configuration Management. In: Proc. of 12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010), Osaka, Japan, Apr 2010.
19. Elbadawi, K., Yu, J.: High Level Abstraction Modeling for Network Configuration Validation. In: Proc. of IEEE Global Telecommunications Conference (GLOBECOM 2010), Miami, USA, Dec 2010.

Paper D: An Adaptable Capability Monitoring System

Patcharee Thongtra and Finn Arve Aagesen

Published in
Proceedings of the IARIA/IEEE 6th International Conference on Networking and Services (ICNS 2010)

Cancun, Mexico, March 2010

An Adaptable Capability Monitoring System

Patcharee Thongtra
Department of Telematics
Norwegian University of Science and Technology
N-7491 Trondheim, Norway
patt@item.ntnu.no

Finn Arve Aagesen
Department of Telematics
Norwegian University of Science and Technology
N-7491 Trondheim, Norway
finnarve@item.ntnu.no

Abstract—SNMP-based monitoring systems are founded on a platform with precompiled agents in the managed node. Software updates and re-configuration are not flexibly handled. This paper presents a solution for monitoring by use of a service framework for adaptable service systems in cooperation with traditional SNMP agents. A two-level manager system consisting of Main monitoring managers and Intermediate monitoring managers executes on TAPAS (Telematics Architecture for Play-based Adaptable Systems) platform for adaptable service systems. The Intermediate monitoring managers are federated managers that communicate with ordinary precompiled SNMP agents by using SNMP protocol. The manager part of the monitoring system will have flexibility both with respect to software updates and re-configuration. The managers are TAPAS role figures. Role figures can be deployed, instantiated and moved during run-time. This feature can be used in overload and fault situations. Some experimental results are presented for cases where the Main monitoring manager is autonomously moved after failure.

Keywords-Network Management, Decentrailzed Monitoring System, Adaptable Service System, EFSM-based Specification

I. Introduction

Network monitoring systems are traditionally based on a centralized model, where a central manager collects, aggregates and processes data retrieved from management agents. Simple Network Management Protocol (SNMP) [1], widely accepted as standard network management protocol, is designed according to this centralized model. Several improvements of the centralized solutions have been proposed. Examples are Remote Network Monitoring (RMON) [2], Management by Delegation (MbD) model [3], [4] as well as a distributed architecture and several Management Information Bases (MIBs) from the IETF Distributed Management (DISMAN) Working Group (WG) [5].

This paper also presents a solution for capability monitoring based on two levels of monitoring managers: *Main* monitoring managers and *Intermediate* monitoring managers. Intermediate monitoring managers are federated managers that communicate with ordinary precompiled SNMP agents. Main monitoring managers distribute monitoring tasks to the Intermediate monitoring managers. We will, however, further enhance the flexibility by the following added features:

- Adaptability in case of overload and fault situations during the run-time
- Update of software implementing the management functionality during the run-time

The added features are attained by using *TAPAS service framework* [6] for the specification and execution of the managers. The main contributions in this paper are:

- the presentation of service framework (TAPAS) that provides a rich set of adaptability and flexibility features to *service system* constituted by distributed Extended Finite State Machine (EFSM)-based software components.
- the illustration how this service framework can be used to attain the adaptability and flexibility features as defined above in a distributed capability monitoring system

Capability is here defined as an inherent property of a node used as a basis to implement service [7]. Capabilities are classified according to capability types and capability parameters. A capability type can be physical resources, software, protocols and data. Capability parameter defines the characteristics of a capability type. The monitoring system presented has capabilities both represented in TAPAS capability ontology as well as in the SNMP MIB type system. A conversion is needed between these two ontology systems. The presentation of the ontology as well as the conversion functionality is not within the scope of the paper.

The rest of the paper is organized as follows. Section II reviews related work. Section III presents TAPAS service framework. Section IV presents the monitoring system. A behavior example demonstrating the flexibility features is presented. Some experimental results for cases where the Main monitoring manager fails are also presented. Section V gives summary and conclusions.

II. Related Work

As mentioned in Sec. I our proposed solutions are related to three research issues: *decentralization, adaptability* and *software update flexibility*.

A. Decentralization

Some solutions are mentioned already in Sec. I. In Management by Delegation (MbD) model [3], [4], they use mobile-code technology to delegate the management functionality described in executable code to cooperating managers during run-time. This approach was also integrated into SNMP architecture, such as Case and Levi [8] implemented a prototype of the MbD approach entirely based on SNMP, and DISMAN WG [5] proposed Script MIB [9] as the continuation of Case and Levi's prototype and the MbD approach. The Script MIB provides mechanisms to transfer the management functionality by means of scripts, to start, suspend, resume and terminate the scripts. DISMAN WG defined also the distributed architecture, where management functionality is distributed over a hierarchy of several distributed managers. This architecture can be applied to other protocols as well, such as in [10] it is applied to Network Configuration (NETConf) protocol [11].

B. Adaptability

An adaptable service system is here defined as a service system that is able to adapt dynamically to changes to service users, nodes, capabilities, system performance and service functionality. Service frameworks proposed in [12]-[15] as well as TAPAS [6] are applicable for service system in general. Mukhija and Glinz [12] have presented a framework that monitors the services and adapts the service behaviors according to a pre-defined XML-based adaptation policy. In DySOA project [13], they monitor the information regarding a set of predefined QoS parameters. The collected information is checked against expected values, and in case of a deviation the re-configuration is triggered. They use a variability model [16] to describe the configuration of a system as a set of variants bound to all variation points. Garlan and Schmerl [14] have presented adaptation framework for self-healing systems. The adaptation is supported by the use of software architectural models [17], which the overall structure of an executing system is captured as a composition of interacting components. Pre-defined operations of the components will be activated to adapt the system. DACIA framework [15] uses mobile-code technology to create adaptable collaborative systems. Similar to our approach, a state-based component in DACIA can move between nodes, while maintaining its state. A state consists of data and messages in the queue, in which the message is used to communicate between the components.

For the works above [12]-[15], it is only focused on some of TAPAS functionalities (see Sec. III.B). For TAPAS adaptation is triggered by three cases: 1) mismatch between the required system performance and the inherent system performance, 2) faults and 3) service component movements. Case 1) and 2) are used in [12], [13], case 2) is used in [14], and case 1) and 3) are used in [15].

C. Software Update Flexibility

To enable the update of software implementing functionalities that can be changed during the run-time, the functionalities should be defined externally by means of specifications. Also, those specifications are assigned later to *generic execution software component* that can execute the specifications. In TAPAS [6], the functionalities are realized by state machine behaviors, so that, they are defined by state machine specification (EFSM-based specification). *An Actor*, which is a generic execution software component, can download and execute the EFSM-based specification at any time instant. Another approach is to use policies to define the system functionality and to update the policy changes during run-time. An example policy-based framework is the IETF Policy Framework [18]. When using EFSM-based specification*, update* of changes in service components are done by deployment of the whole EFSM-based service component specification. When using policies only incremental changes are deployed. But the complete policy based functionality need to be validated off-line before the deployment of the incremental changes. To provide more flexibility, one work [19] in the TAPAS project combines EFSM-based specification with the policies. In this paper we have focus on only EFSM-based service specifications.

III. TAPAS Service Framework

TAPAS service framework consists of architecture and platform. The architecture consists of *a computing architecture* and *a service functionality architecture*.

The service functionality architecture defines the structure and content of service functionalities independent of implementation. The computing architecture has focus on the modeling of functionality with respect to implementation. TAPAS platform is a support system for software deployment, execution and management of service systems defined by the architecture concepts.

A. TAPAS Computing Architecture

The computing architecture is illustrated in Fig. 1. For simplicity, the service user and service level agreements (SLA) are not included in the figure. The computing architecture has three views: the *service* view, *play* view and *physical* view. The service view considers the service as constituted by service components. The leaf 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.

The computing architecture is founded on a theatre metaphor. Actors perform roles according to manuscripts that can be downloaded. A service system is constituted by a play consisting of several actors playing different roles. A *Director* is an actor that controls and/or supports other actors in a play. For example, it supports an actor in the process of becoming a role figure.

The EFSM functionality constituting the service components are TAPAS role figures which further are realized by TAPAS actors. Actors are generic EFSMs that can constitute role figures by downloading and executing behavior specifications. Role figures are deployed and instantiated during run-time in nodes with sufficient *capabilities*. This feature can be used to move service components during run-time in case of overload or fault situations.

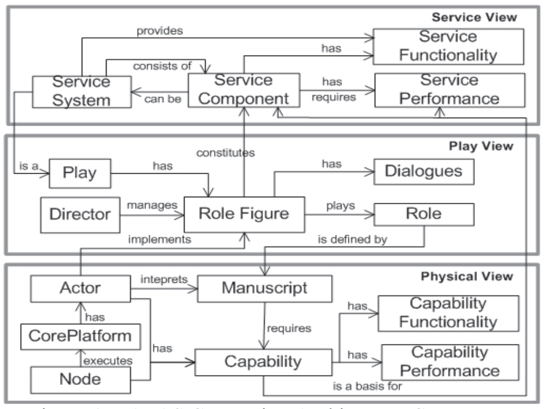


Figure 1. TAPAS Computing Architecture Concepts.

Capability is defined already in Sec. I. Considering the service component they have *service parameters* that also can be classified as functional or performance parameters in the same way as for capabilities.

The manuscript can be based on an Extended Finite State Machine (EFSM) or a Reasoning Machine (RM) model [19]. In this paper, only the EFSM manuscript is used. The EFSM model used is defined as:

$$E \equiv \{ S_M, S_I, S_S, V, M, O, Q, F_S, F_O, F_V\}$$

where S_M is a set of states, S_I is an initial state and S_S is a set of *stable* states. V is a set of variables. M is a set of input messages, O is a set of output messages, Q is a message input queue, F_S is state transition function (F_S: S x M x V -> S), F_O is output function (F_O: S x M x V -> O) and F_V is set of functions. During the execution, a service component maintains dynamic *EFSM data* that is composed of current state, current variables and messages in the queue.

A *stable state* is a state where a service component can move safely, i.e. a state when a service component can be re-instantiated in a new node and restore state, variables and queued messages from the EFSM data. An important variable is the *dialogue variable*. This variable identifies an instance of a *dialogue_object* that defines the role figure itself and its cooperating role figures as follows:

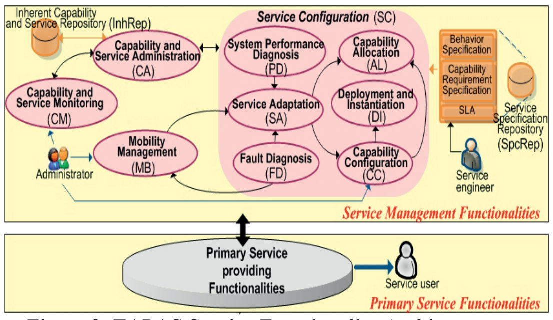
dialogue_object = {role_figure_id},
role_figure_id = (service_component_id, actor_id),
service_component_id = (play_id, role_id),
actor_id = (node_id, actor_number).

The identifier node_id is defined by the IP address of the node. A local numbering scheme is used for actor_number.

B. TAPAS Service Functionality Architecture

The service functionality architecture is illustrated in Fig. 2. The primary service functionalities provide services to the service users. The service management functionalities include management of services as well as network components. Note that the figure illustrates a *generic functional model.* A complete specification will comprise:

- An functional organization model, i.e. a model which reflects how the functionality of the service components are further refined in cooperating sub components
- A physical model, i.e. the distribution of the organization model components in physical nodes.



Inherent Capability and Service Repository (InhRep)
Service Configuration (SC)
Capability and Service Administration (CA)
System Performance Diagnosis (PD)
Capability Allocation (AL)
Behavior Specification
Capability Requirement Specification
SLA
Capability and Service Monitoring (CM)
Service Adaptation (SA)
Deployment and Instantiation (DI)
Service Specification Repository (SpcRep)
Administrator
Mobility Management (MB)
Fault Diagnosis (FD)
Capability Configuration (CC)
Service engineer
Service Management Functionalities
Primary Service providing Functionalities
Service user
Primary Service Functionalities

Figure 2. TAPAS Service Functionality Architecture.

The service management functionality components are:

- *Service Specification Repository (SpcRep)*

- *Inherent Capability and Service Repository (InhRep)*
- *Capability and Service Administration (CA)*
- *Capability and Service Monitoring (CM)*
- *Mobility Management (MB)*
- *Service Configuration (SC)*

Service Specification Repository stores role figure behavior specifications, capability type and capability parameter requirements specifications, and SLAs. *Inherent Capability and Service Repository* stores data about available nodes, capabilities and instantiated services.

Capability and Service Administration handles the registration of nodes, capabilities and instantiated services. This component also provides a current view of available nodes, capabilities and instantiated services.

Capability and Service Monitoring monitors nodes, capabilities and services. This component gets monitoring requests from CA as well as updates monitored data via CA.

Mobility Management handles various mobility types such as personal mobility, terminal mobility as well as service component mobility.

Service Configuration has the sub-components as illustrated in the figure. *Capability Configuration (CC)* finds and selects target nodes that satisfy the capability requirements of the service. *Capability Allocation (AL)* allocates capabilities to the service users of the various service classes. This task is performed in accordance with the requirements of *system performance*, here defined as the sum of capability performance and service performance, defined in the SLA. The component *Deployment and Instantiation (DI)* comprises *deployment* which is the introduction of new service components in nodes, and *instantiation* which starts execution of deployed service components. *Fault Diagnosis (FD)* detects faults. *System Performance Diagnosis (PD)* detects mismatch between the required system performance and the inherent system performance. *Service Adaptation (SA)* plans the adaptation during the service system execution in case of 1) mismatch between the required system performance and the inherent system performance, 2) faults and 3) other reasons for service component movements. The adaptation is based on AL only, or the combination of CC and AL.

The *service engineer* uploads the specifications and SLA into SpcRep. The *administrator* starts new services by the use of Capability Configuration. The administrator also has a privilege to monitor nodes and services as well as to move service components by the use of Capability and Service Monitoring and Mobility Management respectively.

Solutions for CC, DI, MB and PD are discussed in [7], [20], [21] and [19]. Solutions for AL will depend on the system performance requirements defined in the SLA, the capability structure as well as the dimensioning and optimization criteria applied by the service provider.

C. TAPAS Platform and Messages

TAPAS platform consists of core platform and management platform. Core platform supports the functionality in TAPAS computing architecture. Management platform supports the functionality defined in the TAPAS service functionality architecture. TAPAS platform execution functionality is based on Java. The behavior specification is XML-based, while the capability requirement specification and SLA are based on OWL [22] and RDF [23] respectively.

TAPAS platform has a *set of procedures* and a *set of messages* intended to be used by the service engineers. The procedures are meant to be high level user friendly interfaces and can be replaced by use of the messages that need more internal insight to the inner architecture. Here only the messages are presented. The messages are classified into six groups:

- *General request*
- *Trouble report*
- *Mobility request and report*

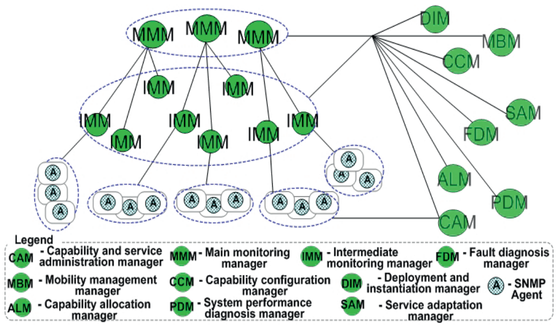
- *Diagnosis*
- *Configuration*
- *Capability Monitoring*

All messages from these groups are listed in Appendix. Some of the messages will be in the behavior description of the monitoring system in Sec. IV.B.

IV. The Monitoring System

A. The Organization Model

The organization of the monitoring system is illustrated in Fig. 3. SNMP Agents execute in managed nodes, while the Main monitoring managers (MMM) and Intermediate monitoring managers (IMM) execute in nodes with TAPAS platform. With reference to the service functionality architecture presented in Sec. III.B, the main task of the monitoring system is *Capability and Service Monitoring (CM).* This functionality is, however, closely related to Capability and Service Administration (CA).


MMM
IMM
DIM
MBM
CCM
SAM
FDM
ALM
PDM
CAM
A
Legend
CAM - Capability and service administration manager
MMM - Main monitoring manager
IMM - Intermediate monitoring manager
FDM - Fault diagnosis manager
MBM - Mobility management manager
CCM - Capability configuration manager
DIM - Deployment and instantiation manager
A - SNMP Agent
ALM - Capability allocation manager
PDM - System performance diagnosis manager
SAM - Service adaptation manager
Figure 3. Organization Model of the Monitoring System.

The *monitored parameters* for a capability is defined by the following objects:
cp_mon_object = (cp_type,cp_param_list),
where
cp_param_list = {cp_param,cp_param_value}.

An IMM is responsible for a *monitoring domain* consisting of a number of nodes. The monitored *capability parameters* related to a node, a domain and Main monitoring manager are defined as follows:

node_mon_object=(node_id, {cp_mon_object}),
domain_mon_object = {node_mon_object},
mmm_mon_object={(imm_id,
domain_mon_object)},

where imm_id is *role_figure_id* of an IMM. As already stated in Sec. I, functionality is needed that converts between the parameters of capabilities represented by TAPAS capability ontology and SNMP capability parameters that are defined by using SNMP MIB types. The presentation of this functionality is not within the scope of the paper. For the realization of adaptability features of the monitoring system, the following service management functionalities are needed:

- Mobility Management (MB)
- Service Configuration with the sub functions: Capability Configuration (CC), System Performance Diagnosis (PD), Fault Diagnosis (FD), Service Adaptation (SA), Capability Allocation (AL), and Deployment and Instantiation (DI).

To realize these functionalities, the model has dedicated role figures in addition to the monitoring managers. These role figures will partly or completely implement the functionality components. These role figures are also denoted as *managers*. Capability allocation manager (ALM) implements part of Capability Allocation functionality, Fault diagnosis manager (FDM) implements part of the Fault Diagnosis functionality, and Capability and service administration manager (CAM) implements part of Capability and Service Administration functionality. CAM regularly discovers new nodes by sending SNMP messages to the agents in such nodes. CAM also registers the new nodes as available nodes and requests MMM to monitor them.

The Fault Diagnosis functionality is distributed between MMM, IMM and FDM. *HeartBeat* messages are used to indicate that a cooperating role figure is alive. Role figures keep track of a cooperating role figure's last *HeartBeat* message. If the time from last

HeartBeat is longer than a defined interval, the role figure sends a *HeartBeatLost* message to FDM. Then FDM will send a *ConfirmAlive* message to the role figure, from which a *HeartBeat* message was missing. *ConfirmAlive* has a high priority. It is used to indicate whether a role figure fails or it is too overloaded and delays. When a role figure gets a *ConfirmAlive*, it must reply the sender with *Alive* message immediately. The use of Heartbeat or I-am-live message is commonly found in failure detection systems [24]-[26].

B. Monitoring system behavior example

A behavior example is given to illustrate some aspects of the flexibility of the monitoring system. In this example, capabilities are not included. The selection of nodes is based on a list of potential nodes rather than defined required capability types and capability parameters. Accordingly, System Performance Diagnosis and Capability Allocation are not included. The behavior example illustrates the behavior for the following six functions:

I): The service engineer uploads MMM- and IMM specifications to SpcRep. The monitoring system is deployed and instantiated during run-time.

II): Managers exchange Heartbeat messages.

III): MMM fails. The failed manager is autonomous moved.

IV): IMM fails. The failed manager is autonomous moved.

V): The service engineer uploads the new version of MMM- or IMM specification to SpcRep. The new versions are deployed and instantiated during run-time.

VI): The administrator moves MMM or IMM during run-time.

Fig. 4 and 5 illustrate the function I, II and III. Fig. 4 illustrates the sequence of messages exchanged between Actors and SNMP Agents located in physical nodes. Fig. 5 gives a time sequence diagram that illustrates the same exchange of messages as in Fig. 4. All messages used are listed in Appendix. The *abbreviations*: start_srvcom_req, start_srvcom_result, serv_adpt, serv_adpt_result, rf_mov, inform_DL, set_DL, get_EFSM, set_EFSM, RF_fail, RF_spen, RF_ctn, HB, HB_lost, deploy, deploy_result, role_assign, mon_cp, update_monitored_cp, inform_mon _domain, set_mon_domain and set_mmm_mon_object, are related to the following *full name* messages respectively: StartServiceComponentRequest, StartServiceComponentResult, ServiceAdaptationRequest, ServiceAdaptationResult, RoleFigureMoveRequest, InformDialogue, SetDialogue, GetEFSMData, SetEFSMData, RoleFigureFailed, RoleFigureSuspended, RoleFigureContinued, HeartBeat, HeartBeatLost, DeployAndInstantiateRF, DeployAnd-InstantiateRFResult, RoleAssigning, MonitorCapability-Request, UpdateMonitoredCapability, InformMonitoring-Domain, SetMonitoringDomain and SetMonitoringDomainsUnderMainManager.

The behavior of the functions I)-VI) are explained below:

Function I):

1) MMM_1 gets a mon_cp request to monitor capability parameters defined by one or more instances of *cp_mon_object*. It distributes the request to an IMM.

1.1) If there is no available IMM, MMM_1 requests CCM to search for a potential node and further to deploy and instantiate a new IMM.

2) When an IMM gets a request distributed from MMM_1, the parameters of *cp_mon_object* are transformed to Object Identifiers (OIDs) of SNMP managed objects. IMM queries OIDs from the SNMP Agent.

3) An IMM updates a set of the resulting *cp_mon_object* to CAM which is responsible for updating InhRep.

3.1) An IMM will update the *cp_mon_object* to MMM_1 only when there is an uncommon change to the measured values.

Function II):
4) HBs are sent between MMM and IMMs every heartbeat interval (T_{HB}).

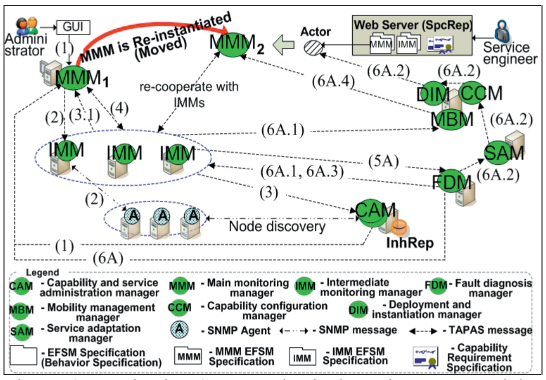
Function III):
5A) An IMM detects the HB from MMM_1 is lost. It suspects MMM_1 failed, and sends a HB_lost message to FDM.
6A) FDM investigates by sending a ConfirmAlive message to MMM_1. If MMM_1 replies by Alive message, then FDM does nothing more. If MMM_1 still have no response or it cannot be reached, then:

6A.1) FDM sends a RF_fail message to broadcast about the MMM_1 failure. As a result, every IMM is aware of the failure, and it sends an inform_DL message and an inform_mon_domain message to inform MBM about a *dialogue_object*, its *role_figure_id* and *domain_mon_object*.

6A.2) FDM requests SAM to plan the adaptation. Then, SAM asks CCM to search for a potential node and further to re-deploy and re-instantiate MMM. As a result, a new MMM (MMM_2) is instantiated.

6A.3) FDM sends an RF_ctn message to broadcast about the availability of MMM (it is about the existence of MMM_2, which substitutes MMM_1).

6A.4) MBM sends MMM_2 a set of *dialogue_object* and a *mmm_mon_object* by a set_DL message and a set_mmm_mon_object message respectively. So that, MMM_2 can handle existing IMMs and re-cooperate with them.


GUI
Administrator
(1)
MMM is Re-instantiated (Moved)
MMM1
MMM2
Actor
Web Server (SpcRep)
Service engineer
re-cooperate with IMMs
(6A.4)
(6A.2)
(6A.2)
DIM
CCM
MBM
(6A.2)
(2)
(3.1)
(4)
(6A.1)
IMM
IMM
IMM
SAM
(5A)
(6A.1, 6A.3)
FDM
(6A.2)
(3)
(2)
A
A
A
CAM
Node discovery
InhRep
(1)
(6A)
Legend
CAM - Capability and service administration manager
MMM - Main monitoring manager
IMM - Intermediate monitoring manager
FDM - Fault diagnosis manager
MBM - Mobility management manager
CCM - Capability configuration manager
DIM - Deployment and instantiation manager
SAM - Service adaptation manager
A - SNMP Agent
SNMP message
TAPAS message
EFSM Specification (Behavior Specification)
MMM EFSM Specification
IMM EFSM Specification
Capability Requirement Specification
Figure 4. Monitoring System Physical Deployment Model - Actors and Agents Interactions.

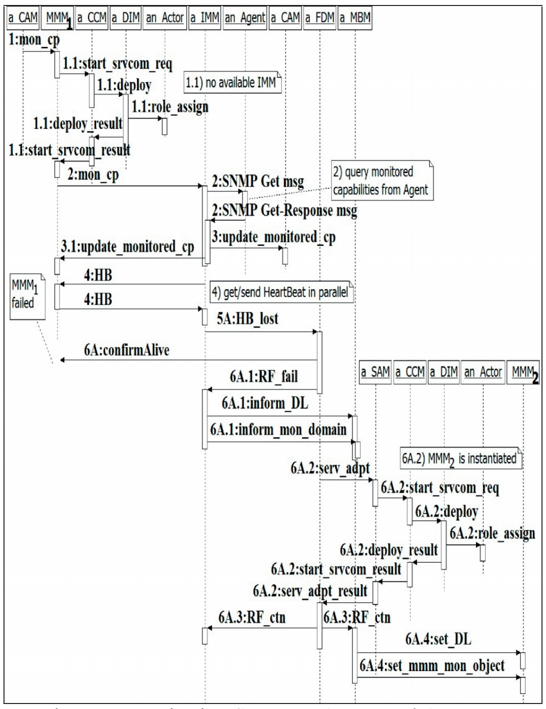


Figure 5. Monitoring System – Actors and Agents Message Sequence Diagram.

Function IV):
5B) An IMM's failure is detected by the HB lost. MMM_1 will send an inform_DL message and an inform_mon_domain message to inform MBM about a *dialogue_object* related to the failed IMM, a *role_figure_id* and a *domain_mon_object* of the failed IMM. A new

IMM (IMM_N) is instantiated. IMM_N is also given the *dialogue_object* and the *domain_mon_object* via a set_DL message and a set_mon_domain message respectively. So that, it can recover the monitoring task as well as re-cooperate with MMM_1.

Function V):

5C) MBM detects that the MMM specification has been changed. MBM sends a get_EFSM message to MMM_1. The message is to request EFSM data: current state, current variables which include the *mmm_mon_object*, and queued messages. After it gets the EFSM data from MMM_1, MBM will send a RF_spen message to broadcast about the MMM_1 suspension, and terminates MMM_1.

6C) A new MMM (MMM_3) is instantiated in the same node as the terminated MMM_1. (The new specification is applied, as it is downloaded and executed by an actor.) MMM_3 will get a set_EFSM message containing the EFSM data. (The EFSM data is maintained during the re-instantiation period). MMM_3 restores state, variables and queued messages from the EFSM data. So that, it can handle existing IMMs and re-cooperate with them.

7C) MBM sends an RF_ctn message to broadcast about the existence of MMM_3.

8C) The IMM specification change is handled the same way as the MMM specification change.

Function VI):

5D) Manager movement is handled similar to the specification change (Function V). In this case, MBM gets a rf_mov request, and MMM or IMM is re-instantiated in a new node instead of current node.

C. Experimental Results

An experiment were carried for the case that a MMM fails and it is autonomous moved. This experiment demonstrates the function I, II and III defined in Sec. IV.B. The number of IMM (N) is 1, 2, 5, 10, 25, 50 and 100. T_{HB} is set = 10s. T_D is a period from which the MMM fails until an IMM detects this failure. T_R is a period from which the MMM fails until a new MMM can re-cooperate with all existing IMMs (It means that a new MMM gets the first HB from all of them). We repeated the experiment 10 times. Fig. 6 shows the average-, maximum- and minimum T_D. Fig. 7 shows the average T_D and average T_R. The difference of T_D and T_R is an overhead of the re-instantiation that a new MMM uses to restore the *mmm_mon_object*.

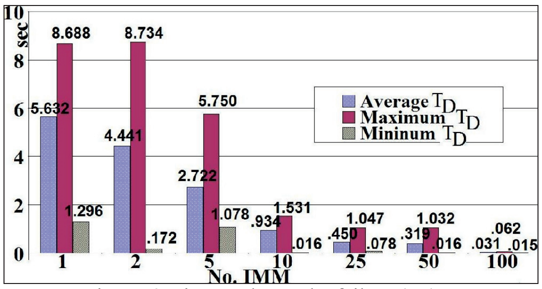


Figure 6. Time to detect the failure (T_D).

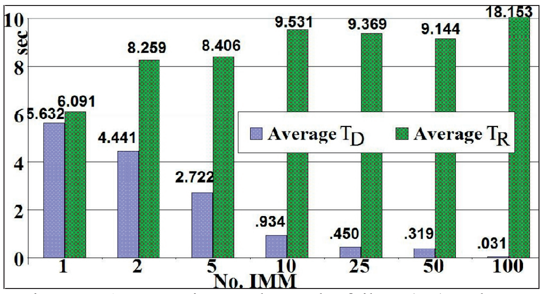


Figure 7. Average time to detect the failure (T_D) and to recover (T_R).

There is an inverse relationship between N and the average T_D. Also, if N is higher, the range of T_D is narrower; the average-, maximum- and minimum T_D are approximately of the same value. T_D can be predicted quite correctly. For example when N = 100, the average T_D is very small (0.031s). The average-, maximum- and minimum T_D are 0.031s, 0.062s and 0.015s respectively. But if N is above a certain limit, the average T_R and the overhead increase sharply. They are notably large (18.153s and 18.122s), when N = 100. This is because it is time-consuming to restore the *mmm_mon_object* with large size and to keep track of HBs from 100 IMMs.

V. Conclusions

A flexible solution for monitoring by use of the TAPAS service framework for adaptable systems has been presented. The monitoring function is performed by two levels of monitoring managers and SNMP agents. Main monitoring managers control Intermediate monitoring managers that communicate with ordinary SNMP agents. Flexibility is attained by using TAPAS service framework for the monitoring managers. The management functionality is EFSM-based. The managers are initially deployed and instantiated in suitable nodes. They are movable during run-time. Fault and system performance diagnosis functionality makes autonomous adaptation possible in case of overload or fault situations. A behavior example demonstrating the software update and adaptability features is presented. Some experimental results are also presented. These results comprise cases where the Main monitoring manager fails. The functionality of the failed manager is then moved and re-instantiated, before the co-operation with the Intermediate monitoring manager is recovered. Indeed, the TAPAS service framework solution applied for the monitoring system can be applied for distributed service systems in general.

References

[1] W. Stallings, *SNMP, SNMPv2, SNMPv3, RMON 1 and 2*. The 3rd edition, Addison-Wesley, 1999.

[2] S. Waldbusser, "Remote Network Monitoring Management Information Base," RFC 1271, November 1991.

[3] Y. Yemini, G. Goldszmidt and S. Yemini, "Network management by delegation," in *Proc. of International Symposium on Integrated Network Management*, 1991.

[4] G. Goldszmidt and Y. Yemini, "Distributed management by delegation," in 15th *International Conference on Distributed Computing*, 1995.

[5] DISMAN - The Distributed Management Working Group. Available at: http://www.ietf.org/wg/concluded/disman.html.

[6] TAPAS: Telematics Architecture for Play-based Adaptable Systems. Available at: http://tapas.item.ntnu.no.

[7] F. A. Aagesen, P. Supadulchai, C. Anutariya and M. M. Shiaa, "Configuration Management for an Adaptable Service System", in *Proc. of IFIP International Conference on Metropolitan Area Networks, Architecture, Protocols, Control, and Management*, Viet Nam, April 2005.

[8] J. D. Case and D. B. Levi, "SNMP mid-level-manager MIB," Internet Draft, SNMP Research, Inc., 1993.

[9] D. Levi and J. Schonwalder, "Definitions of Managed Objects for the Delegation of Management Scripts," RFC 3165, Aug 2001.

[10] N. Carrilho and N. Ventura, "Distributed Policy-based Network Management with NETCONF," in *Proc. of Southern African Telecommunications and Applications Conference (SATNAC 2006)*, September, 2006.

[11] E. R. Enns, "NETCONF Configuration Protocol," Internet draft, RFC 4741, December 2006.

[12] A. Mukhija and M. Glinz, "The CASA Approach to Autonomic Applications," in *Proc. of the 5th IEEE Workshop on Applications and Services in Wireless Networks (ASWN 2005)*, France, June-July 2005.

[13] J. Siljee, I. Bosloper, J. Nijhuis and D. Hammer, "DySOA: Making service systems self-adaptive," in *Proc. of the 3rd International Conference on Service Oriented Computing (ICSOC05)*, Amsterdam, 2005.

[14] D. Garlan and B. Schmerl, "Model-based adaptation for self-healing systems," in *Proc. of the 1st Workshop on Self-Healing Systems*, November 2002.

[15] R. Litiu and A. Prakash, Radu Litiu and Atul Prakash, "Developing Adaptive Groupware Applications using a Mobile Component Framework," in *Proc. of CSCW'2000*, December 2000.

[16] M. Sinnema, S. Deelstra, J. Nijhuis and J. Bosch, "COVAMOF: A Framework for Modeling Variability in Software Product Families," in *the 3rd Software Product Line Conference (SPLC 2004)*, USA, 2004.

[17] M. Shaw and D. Garlan, *Software Architectures: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[18] R. Yavatkar, D. Pendarakis, and R. Guerin, "A Framework for Policy-Based Admission Control," RFC 2753, January 2000.

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

[20] S. Jiang and F. A. Aagesen, "Xml-based dynamic service behaviour representation," in *NIK'2003*, Norway, November 2003.

[21] M. M. Shiaa, "Mobility Management in Adaptable Service Systems," *Doctoral Thesis, Department of Telematics, NTNU*, Trondheim, 2005.
[22] OWL: Web Ontology Language, Available at: http://www.w3.org/TR/owl-features/.
[23] W3C, "Resource Description Framework (RDF) / XML Syntax Specification", 2004, Available at: http://www.w3.org/TR/rdf-syntax-grammar/.
[24] R. Macedo, "Failure detection in asynchronous distributed systems," in *the 2nd Test and Fault-Tolerance Workshop*, Brazil, 2000.
[25] I. Sotoma and E. R. M. Madeira, "Adaptation - algorithms to adaptive fault monitoring and their implementation on cobra," in *the 3rd International Symposium on Distributed Objects and Applications*, USA, 2001.
[26] F. Lima and R. Macedo, "Adapting Failure Detectors to Communication Network Load Fluctuations Using SNMP and Artificial Neural Nets," in *the 2nd Latin-American Symposium on Dependable Computing (LADC '05)*, Brazil, 2005.

APPENDIX

A message contains *message ID*, *sender ID*, *receiver(s) ID*, *system time for composing the message* and *message data*. Message data is message name and message parameters. The message ID is a unique identifier for a message. The sender ID and receiver(s) ID are an *actor_id* and a set of *actor_id* respectively. All messages are listed as follows in the form:
MessageName:*MessageParameters*:

General request message group
StartServiceComponentRequest:*service_component_id, spec_location*
StartServiceComponentResult:*role_figure_id*
StopServiceComponentRequest:*service_component_id,role_figure_id*
ServiceAdaptationRequest:*{service_component_id}*
ServiceAdaptationResult:*{new_role_figure_id}*
NodeRegistration:*node_id*
PrimaryServiceFunctionalityRequest: *data_on_purpose_of_functionality*

Trouble report message group
RoleFigureUnreachable:*role_figure_id*
InsufficientCapability:*actor_id*
QoSDegradation:*{service_component_id}*
NodeNoResponse:*node_id*

Mobility request and report message group
RoleFigureMoveRequest:*role_figure_id,new_node_id*
InformDialogue:*dialogue_object*
SetDialogue:*{dialogue_object}*
GetEFSMData:
EFSMData:*curr_state,curr_vars,curr_msgs*
SetEFSMData:*curr_state,curr_vars,curr_msgs*

Diagnosis message group
RoleFigureFailed:*role_figure_id*
RoleFigureSuspended:*role_figure_id*
RoleFigureContinued:*old_role_figure_id, new_role_figure_id*
HeartBeat:
HeartBeatLost:*role_figure_id*
ConfirmAlive:
Alive:

Configuration message group
GetNodeCapabilitiesRequest:*node_id*
NodeCapabilities:*{cp_mon_object}*
SetNodeCapabilitiesRequest:*node_mon_object* **GetCapableNode:***{cp_mon_object}*
CapableNode:*{node_id}*
DeployAndInstantiateRF: *service_component_id,spec_location,node_id*
DeployAndInstantiateRFResult:*role_figure_id*
RoleAssigning:*service_component_id, spec_location*
RoleTermination:*role_figure_id*
ActorTermination:*actor_id*

Capability Monitoring message group
MonitorCapabilityRequest:*node_mon_object, monitoring_interval*
StopMonitorCapabilityRequest:*node_id*
UpdateMonitoredCapability:*node_mon_object*
InformMonitoringDomain:*imm_id,domain_mon_object*
SetMonitoringDomain:*domain_mon_object*
SetMonitoringDomainsUnderMainManager:*mmm_mon_object*

Paper E: Towards Policy-Supported Adaptable Service Systems

Paramai Supadulchai, Finn Arve Aagesen and Patcharee Thongtra

Published in
Proceedings of the 13th Eunice Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services

Twente, Netherlands, June 2007

Lecture Notes in Computer Science (LNCS) 4606, pp. 128-140, 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 0, 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.

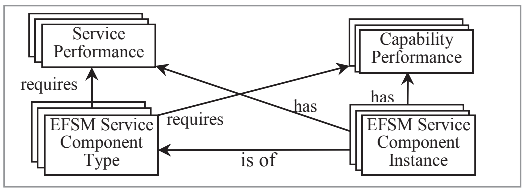


Fig. 1. EFSM part of Service System – Concept Structure

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:

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
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:

$$\mathcal{T} \equiv Expr\,\{S, C, I, (E_k, \hat{S}_{R,k}, \hat{C}_{R,k}; k = [1, K])\} \qquad \textbf{(3)}$$

$$\mathcal{E} \equiv \{((\hat{E}_{lk}, \hat{S}_{I,lk}, \hat{C}_{I,lk}; l = [1, L_k]), k = [1, K]), (\hat{C}_{A,n}; n = [1, N_{Node}])\} \qquad \textbf{(4)}$$

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).

$$\Sigma \equiv Expr\,\{S, C, (E_k, \hat{S}_{R,k}, \hat{C}_{R,k}; k = [1, K])\} \qquad \textbf{(5)}$$

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.

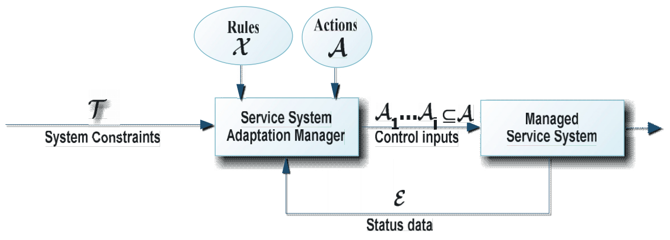


Fig. 2. Policy-based adaptation using static policies

4.3 Policy-based adaptation using dynamic policies

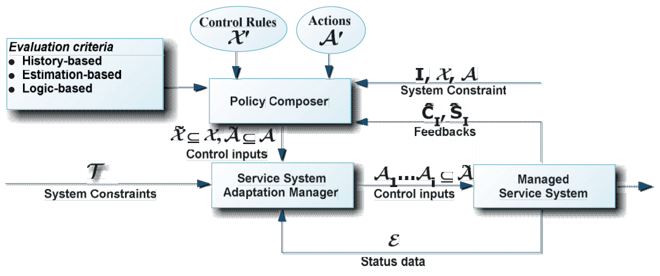


Fig. 3. Policy based adaptation based on dynamic policies

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:

$$\mathcal{R}_1 \equiv \{ \mathcal{Q}, \mathcal{F}, \tilde{\mathcal{P}}, \mathcal{T}, \mathcal{E}, \Sigma \} \quad \textbf{(6a)}$$

$$\tilde{\mathcal{P}} \equiv \{ \tilde{\mathcal{X}}, \tilde{\mathcal{A}} \} \quad \textbf{(6b)}$$

$$\mathcal{R}_2 \equiv \{ \mathcal{Q}', \mathcal{F}, \mathcal{P}', \mathcal{T}', \mathcal{E}', \Sigma' \} \quad \textbf{(6c)}$$

$$\mathcal{P}' \equiv \{ \mathcal{X}', \mathcal{A}' \} \quad \textbf{(6d)}$$

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

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.

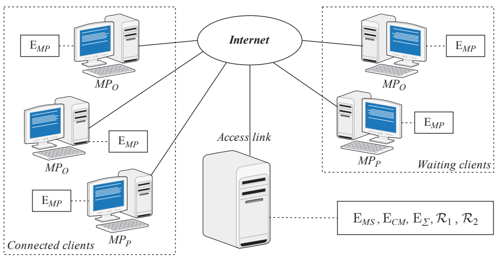


Fig. 4. A music video on-demand service system; E_{MS}: Media server type, E_{MP}: Media player type, E_{CM}: Capability manager type, E_{Σ}: Dedicated EFSM type for controlling the reasoning mechanism, $\mathcal{R}_1$: Service system adaptation manager type, $\mathcal{R}_2$: Policy evaluator type.

The basic EFSM types constituting the capability management system are media server handler (E_{MS}), media player handler (E_{MP}) and capability manager (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:

$$\begin{aligned} m_T = \; & m(MP_O, X_{I,O})\times T_{Serv,O} + m(MP_P, X_{I,P})\times T_{Serv,P} - p_{Wait}(MP_O)\times T_{Wait,O} \\ & - p_{Wait}(MP_P)\times T_{Wait,P} - p_{Dis}(MP_O)\times N_{Dis,O} - p_{Ser}\times(N_{MS}\text{-})\times\Delta \qquad \textbf{(7)} \end{aligned}$$

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 illustrates the accumulated total income when $\rho = 3.45$. 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.

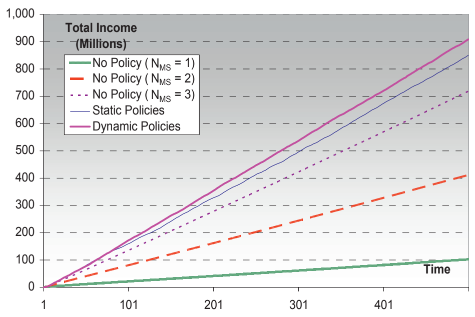


Fig. 5. Accumulated total income for ρ = 3.45

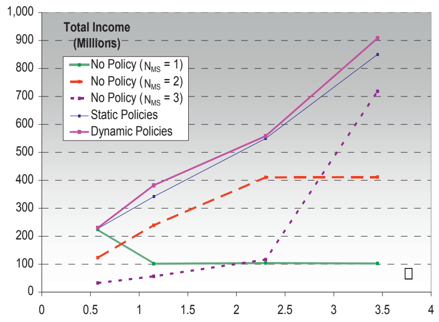


Fig. 6. Accumulated income at 500th ms

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 $10 \times d_{SLA\_Class}$, provides much time for the system for learning the consequences of the rules being applied. Case V gives a better result.

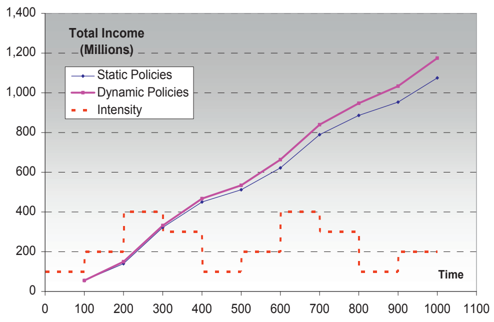


Fig. 7. Accumulated total income

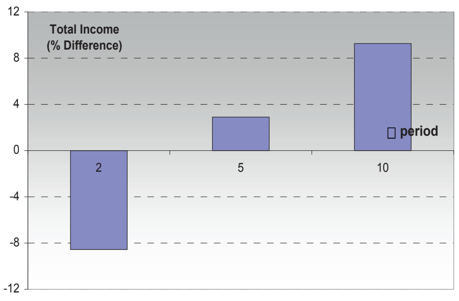


Fig. 8. Comparison of Case IV&V

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× and $5 \times d_{\text{SLA\_Class}}$.

The use of $\mathcal{X}_3$, $\mathcal{X}_4$ (see Appendix), which will add or remove an $\bar{\text{M}}$S, 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:

$$\mathcal{A} \quad \equiv \quad \{ \mathcal{A}_D, \mathcal{A}_B, \mathcal{A}_I, \mathcal{A}_R \} \qquad \textbf{(A.1)}$$

$\mathcal{A}_D$ *(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:

$$\mathcal{X} \quad \equiv \quad \{ \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3, \mathcal{X}_4 \} \qquad \textbf{(A.2)}$$

$\mathcal{X}_1$ 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 $\Sigma_{T2} \doteq \Sigma_{T1}$ ($\doteq$ means '*is instantiated as*'), and goal condition $\Sigma_{G2} \doteq \Sigma_{G1}$. The set of actions $\mathcal{A}$ applied by the the policy evaluator is:

$$\mathcal{A}' \quad \equiv \quad \{\mathcal{A}_G(\mathcal{X}_i), \mathcal{A}_T(\mathcal{X}_i) \} \qquad \textbf{(A.3)}$$

$\mathcal{A}_G(\mathcal{X}_i)$ 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:

$$\mathrm{Qo}\mathcal{X}_I \quad = \quad \mathrm{Qo}\mathcal{X}_i + \frac{m_{T,t} - m_{T,t-1}}{m_{T,t}} \times 100 \qquad \textbf{(A.4)}$$

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:

$$\mathcal{X}'' \quad \equiv \quad \{ \mathcal{X}''_1, \mathcal{X}''_2 \} \qquad \textbf{(A.5)}$$

$\mathcal{X}''_1$ 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 F: An Autonomic Framework for Service Configuration

Patcharee Thongtra and Finn Arve Aagesen

Published in
Proceedings of the IARIA/IEEE 6th International Multi-Conference on Computing in the Global Information Technology (ICCGI 2011)

Luxembourg, June 2011

An Autonomic Framework for Service Configuration

Patcharee Thongtra
Department of Telematics
Norwegian University of Science and Technology
N-7491 Trondheim, Norway
patt@item.ntnu.no

Finn Arve Aagesen
Department of Telematics
Norwegian University of Science and Technology
N-7491 Trondheim, Norway
finnarve@item.ntnu.no

Abstract—An autonomic framework for service configuration functionality is proposed. The framework has goals and policies. *Goals* express required performance and income measures. *Policies* define actions in states with unwanted performance and income measures. All functionality is executed by *autonomic elements* (AEs) that have ability to download and execute behavior specifications during run-time. An AE has several generic functionality components. Two important generic components of an AE are *Judge* and *Strategist*. A *Strategist* selects actions in a state with unwanted performance and income measures to reach a state defined by goal performance and income measures. A *Judge* gives rewards to actions based on the ability to move towards a state with goal performance and income measures. The *Strategist*'s selection of actions is based on the rewards given by the *Judge*. AE functionality is realized by the combination of Extended Finite State Machines (EFSM), a Reasoning Machine (RM) and a Learning Machine (LM). A case study of an adaptable streaming system is presented. Using the proposed model, the streaming system can select actions for capability allocation adaptation more appropriately as evaluated by the performance and income measure results.

Keywords-Autonomic; Service configuration; Policy; Autonomic Elements.

I. 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 service users [1]. 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. *Service configuration* comprises capability configuration, capability allocation, service deployment and instantiation, system performance diagnosis, fault diagnosis and service adaptation. *A capability* is an inherent property of a node required as a basis to implement services [1]. 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 service configuration is done with respect to required capabilities and capability performances as well as required service performances. In this paper, we focus on service configuration of *adaptable service systems*, here defined as a service system that can adapt by itself related to changes by users, nodes, capabilities, system performances and service functionalities.

In this paper, an *autonomic* approach to adaptable service systems is proposed. *Autonomic systems* have ability to manage themselves and to adapt dynamically to changes in accordance with given objectives [2, 3]. The autonomic system is *constituted* by distributed components denoted as *autonomic elements (AEs)*. An AE is the smallest entity that can manage its internal behaviors and relationships with other entities in accordance with its defined behavior. *A service component as already defined is realized by one AE*. An AE is constituted by several generic functionality components. Two important components are *Judge* and *Strategist*. The *Judge* and *Strategist* apply defined *goals* and *policies*. Goals express required performance and income measures. A policy is defined by conditions, constraints and actions, and defines accordingly actions to adapt the system in states with unwanted performance and income measures. The *Judge* gives rewards to actions based on the ability to move towards a state with goal performance and income measures. The *Strategist* selects actions based on the rewards given by the *Judge*.

The reasons behind the Autonomic Element model and the AE functionality components' specifications (see Section III) are service components based on the classical Extended Finite State Machine (EFSM) approach can provides the software update flexibility [4], and Reasoning Machine (RM) using the policies can add the ability to cope with various situations more flexible [5, 6].

This paper is organized as follows. Section II defines autonomic properties. Section III defines the main concepts of what is denoted as the *Goal-based Policy Ontology*. The details of the *Autonomic Element Model* are described in Section IV. Section V describes how AEs are used to realize service functionalities that are necessary for the service configuration. Section VI presents a case study, related works are presented in Section VII, and finally, summary and conclusions are presented in Section VIII.

II. Properties of autonomic elements

An autonomic system consists of a set of decentralized autonomic elements (AEs), as defined in Section I. AE functionality is realized by the combination of Extended Finite State Machines (EFSM), a Reasoning Machine (RM) and a Learning Machine (LM). An AE is a generic software component that can dynamically download and execute EFSM, RM and LM specifications. Properties can be classified as *individual* AE properties and *shared* AE properties, i.e., properties of the AEs constituted by the cooperation of AEs. AEs have the following *individual* properties:

- *Automaticity*: An AE can manage its EFSM states, variables, actions and policies.
- *Awareness*: An AE is able to monitor its own EFSM states and performances.
- Goal-driven: An AE operates and/or controls its functions towards goals.

AEs have the following *shared* properties:

- *Automaticity*: AEs can manage capabilities and nodes, i.e., capabilities and nodes can be added and removed.
- *Adaptability*: The goals, policies and EFSM behavior of an AE can be changed.
- *Awareness*: An AE is able to monitor available nodes and capabilities. Information about EFSM states and performance measures can be made available to other AEs.
- *Mobility:* An AE can move to a new node and resume its operation.

III. Goal-based Policy Ontology

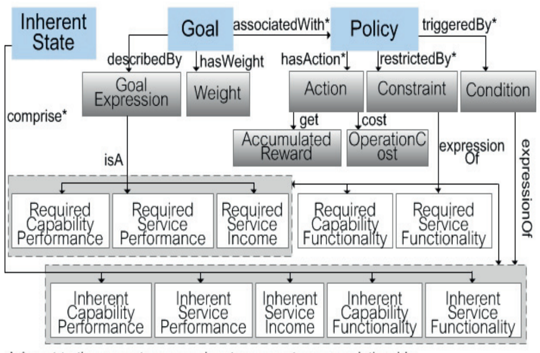
An ontology is a formal and explicit specification of a *shared* conceptualization [7] containing both objects and functions operating on instances of objects. We can define independent concepts and relational concepts defined by mathematical logics, e.g., *if-then-else*. In applications with reasoning capability, the logic concepts can be represented and processed flexibly as rules [1].

Figure 1 presents a simplified diagram of the concepts in the Goal-based Policy Ontology. At the top level we have *goal*, *policy* and *inherent state*, which all are related to service and capability as defined in Section I. The instantiated AEs have inherent states that can comprise *measures* related to functionality and performance of services and capabilities as well as income. *System performance* is defined as the sum of capability performance and service performance.

As a basis for the optimal adaptation, service level agreements (SLA) are needed between the service users and the service provider. An SLA class defines service functionalities, capabilities, QoS levels, prices and penalty. *Service income* includes the estimated income paid by the users for using services in normal QoS conditions and the penalty cost paid back to the users when the service qualities and functionalities are lower than defined by SLA. In general, goal, policy and inherent state concepts have the SLA class as a parameter.

The goal is defined by a *goal expression* and a *weight*. The goal *expression* defines a required system performance or service income measure. A goal example is: "*Service response time of premium service SLA class < 2 secs*". The goal *weight* identifies a goal's importance. A goal can be associated with a set of policies.

A policy is defined by conditions, constraints and actions. The condition defines an inherent state which actions are needed to adapt the service systems. The constraint restricts the usage of the policy, and is described by an expression of required and inherent functionality and performance of services and capabilities, required and inherent service incomes, available nodes and their capabilities, as well as system time. An action has an estimated operation cost and accumulated reward.



A * next to the property names denotes a one-to-many relationship.

Figure 1. Goal-based Policy Ontology.

A policy example related to the goal example given above is: "*If CPU utilization > 95% and the time is between 18:00-24:00, ignore new service requests of users of ordinary SLA classes that request service time > 2 mins*". It is expressed with Conditions: CPU utilization > 95%, Constraints: system time between 18:00-24:00 and service time request > 2 mins, and Actions: ignore new service requests of users of ordinary SLA classes.

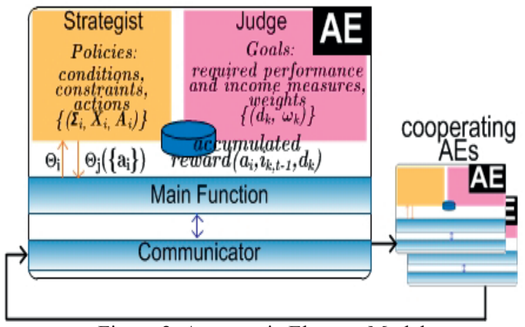
Table I lists notations used for capability, service and income concepts.

Table I. The capability, service and income concept Notation

$\hat{C}_R$	Required capability performance set
$\hat{C}_I$	Inherent capability performance set
$\overline{C_R}$	Required capability functionality set
$\overline{C_I}$	Inherent capability functionality set
$\hat{C}_{A,n}$	Set of available capabilities in node n; n=[1, N]
$\hat{S}_R$	Required service performance set
$\hat{S}_I$	Inherent service performance set
$\overline{S_R}$	Required service functionality set
$\overline{S_I}$	Inherent service functionality set
I_R	Required service income
I_I	Inherent service income

IV. Autonomic Element Model

An AE is composed of four functional modules: i) *Main Function*, ii) *Strategist*, iii) *Judge* and iv) *Communicator*, as illustrated by Figure 2. The behaviors of the various modules are explained in the following subsections IV.A-IV.D. The life-cycle of an AE is described in Section IV.E.



Strategist
Policies: conditions, constraints, actions $\{(\Sigma_i, X_i, A_i)\}$
Judge
Goals: required performance and income measures, weights $\{(d_k, \omega_k)\}$
AE
Θ_i
$\Theta_j(\{a_i\})$
accumulated reward$(a_i, i_{k,t-1}, d_k)$
Main Function
Communicator
cooperating AEs
AE

Figure 2. Autonomic Element Model.

A. Main Function

Main Function coordinates the functionality of an AE. An AE has some general behavior which is common for all AEs, and some behavior depending on the specific role of the AE (see Section V). In general, an AE will have requirements with respect to capabilities and capability functionalities and performances. The specific need depends on the specific functionality of the AE. The *Main Function* behavior is based on an *Extended Finite State Machine (EFSM)* model E defined (≡) as*:*

$$E \equiv \{ S_M, S_I, S_S, V, M, O, Q, F_S, F_O, F_V \} \tag{1}$$

where S_M is a set of all states, S_I is an initial state and S_S is a set of *stable* states. V is a set of variables including the inherent state variables. M is a set of input messages, O is a set of output messages and Q is a message input queue. F_S is a state transition function (F_S: S x M x V -> S), F_O is an output function (F_O: S x M x V -> O) and F_V is a set of actions performed during a specific state transition.

An AE can move to a new node. A *stable state* is a state of the *Main Function* where an AE's functionality can move safely and be re-instantiated in a new node based on the restoration of EFSM state, variables, and queued messages. *Strategist* is used by the *Main Function* to select appropriate actions. The *Main Function* will regularly

- Compare the condition part of the policies with inherent state variables, and will
- Activate the *Strategist* if a condition is met, which returns an action to be used by the *Main Function*

B. Strategist

Strategist selects appropriate actions to be used by the *Main Function*. The *Strategist* behavior is based on a *Reasoning Machine (RM)* model, extended from [5, 6]. This RM is activated by a condition at a time. When it is activated, one of the various policies that have such condition will be selected and its actions will be executed. The RM model R is defined as:

$$R \equiv \{ \Theta, \Phi, \mathcal{P}, \xi \} \tag{2}$$

where Θ is a set of query expressions containing variables, Φ is a generic *reasoning procedure,* $\mathcal{P}$ is a set of policies, and ξ is the strategist data including the inherent states from the *Main Function* and from other AEs, and available nodes and their capabilities.

$$\xi \equiv (\overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I, \hat{C}_{A,n}; n=[1, N]) \tag{3}$$

$$\mathcal{P} \equiv \{ p_i \} \tag{4}$$

$$p_i \equiv (\Sigma_i, X_i, A_i) \tag{5}$$

$$\Sigma_i \equiv \text{Expression}(\overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I) \tag{6}$$

$$X_i \equiv \text{Expression}(\overline{S_R}, \hat{S}_R, \overline{C_R}, \hat{C}_R, I_R, \overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I, \hat{C}_{A,n}; n=[1, N], \Gamma) \tag{7}$$

A policy p_i has conditions Σ_i, constraints X_i and actions A_i. The condition is an expression of the inherent states from the *Main Function* and from other AEs. The constraint is an expression of required functionality and performance of services and capabilities, required service incomes, the inherent states from the *Main Function* and from other AEs, available nodes and their capabilities, as well as system time (Γ).

The reasoning procedure is applied to select appropriate actions with maximum accumulated rewards. It is based on *Equivalent transformation (ET)* [8], which solves a given problem by finding values for the variables of the queries. The conditions, constraints and actions can have variables. The result of the reasoning procedure can, in addition to actions, give instantiated variables.

C. Judge

Judge gives rewards to actions to be selected by the *Strategist*. The reward is a numeric value based on the ability to move towards a state with goal performance and income measures. The rewards will be accumulated over a period of time. The *Judge* behavior is based on a *Learning Machine (LM)* model L defined as:

$$L \equiv \{ \Omega, \Lambda, \Psi, \zeta \} \tag{8}$$

where Ω is a set of *goals*, Λ is a generic *rewarding procedure*, Ψ is a *reward database* storing the accumulated rewards of actions, and ζ is the judge data including the inherent states from the *Main Function* and from other AEs. We further have:

$$\zeta \equiv (\overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I) \quad (9)$$

$$\Omega \equiv \{ g_k \} \quad (10)$$

$$g_k \equiv (d_k, \omega_k) \quad (11)$$

A goal g_k has goal expression d_k and weight ω_k. The sum of the goal weights is equal to 1. At time t, the rewarding procedure will calculate the reward of an action a_i, which was applied at time t-1 as:

$$\mathrm{reward}(a_i, i_{k,t-1}, d_k) = (\Delta(i_{k,t}, i_{k,t-1})/\Delta(d_k, i_{k,t-1})) * \omega_k - \mathrm{cost}(a_i) \quad (12)$$

where $i_{k,t-1}$ and $i_{k,t}$ are an inherent state measure before and after applying the action for an monitoring interval [t-1, t], $i_k \in \zeta$ and d_k is an associated goal required measures. $\Delta(i_{k,t}, i_{k,t-1})$ is the difference between $i_{k,t}$ and $i_{k,t-1}$. $\Delta(d_k, i_{k,t-1})$ is the difference between d_k and $i_{k,t-1}$. ω_k is the goal weight and $\mathrm{cost}(a_i)$ is the operation cost of a_i.

The accumulated reward of an action a_i, $\mathrm{accumulated\_reward}(a_i, i_{k,t-1}, d_k)$, is then the sum of the rewards of a_i, for an inherent state measure $i_{k,t-1}$ and a goal required measure d_k.

D. Communicator

Communicator handles message sending and receiving on behalf of the *Main Function*. The *Communicator* behavior is based on the EFSM model in (1). Other AEs can subscribe to the inherent state variables of an AE. The *Communicator* will manage *subscription* messages and will send inherent state variables to other AEs on behalf of the *Main Function*.
The *Communicator* also handles the registration function on behalf of the *Main Function*. Registration message is sent to *Registry (REG)* (see Section V) that is an important AE that records the life-cycle state of AEs. The registration message contains IP address of the AE.

The *Communicator* will regularly broadcast *heartbeat message*, which is used to indicate that an AE is alive. The heartbeat messages are monitored by *Life Monitor AE (LMO)* (see Section V). In addition, the *Communicator* will inform REG about changes in the life-cycle state of the AE (see Section IV.E). REG will broadcast the changes to other AEs that subscribe to such updates.

E. Autonomic Element Life Cycle

The combined states of an AE during its life-cycle are defined follows:

- *Initial state*: An AE is instantiated in a node where there are capabilities and capability functionalities and performances as required.
- *Registering state*: An AE registers to REG.
- *Normal-Active state*: An AE provides services with normal functionality and QoS level.
- *Degraded-Active state*: If in the Normal-Active-State the capabilities are less than required and results in degraded functionality and QoS, the life-cycle state will change to Degraded-Active state. In this state, some actions selected by the *Strategist* can be taken to upgrade the capabilities and performances. From both the Normal-Active state and the Degraded-Active state, the life-cycle state can change to Moving, Suspended or Terminated.
- *Moving state*: An AE's functionality is being moved and re-instantiated in a new node. A move can only take place if the EFSM states are stable (see Section IV.A).
- *Suspended state*: An AE is suspended, i.e., by an action selected by the *Strategist*, which means that it stops executing current behavior specifications. An AE will release its allocated capabilities. At this state, an AE can start executing new behavior specifications. This makes an AE goes back to the Normal-Active state.
- *Terminated state*: Other AEs detect that an AE's heartbeat message is lost or an AE could not be reached because of some unintentional reasons, e.g., the hardware failure. REG is informed about this by LMO or the other AEs, then REG records that an AE is terminated.

V. Autonomic Element-Based Service Functionality Architecture

The service functionalities required for service configuration are constituted by *AEs* and repositories, as illustrated in Figure 3. The AE responsibilities and the repositories are described below.

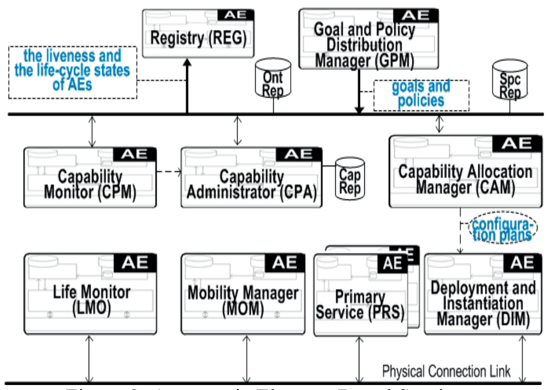

AE
Registry (REG)
Goal and Policy Distribution Manager (GPM)
the liveness and the life-cycle states of AEs
Ont Rep
goals and policies
Spc Rep
Capability Monitor (CPM)
Capability Administrator (CPA)
Cap Rep
Capability Allocation Manager (CAM)
configuration plans
Life Monitor (LMO)
Mobility Manager (MOM)
Primary Service (PRS)
Deployment and Instantiation Manager (DIM)
Physical Connection Link

Figure 3. Autonomic Element-Based Service Functionality Architecture. Solid arrows indicate the physical connections of AEs and dashed arrows represent the message flows between AEs.

- *Primary Service (PRS)* provides ordinary user services.
- *Registry (REG)*, as already mentioned in Section IV.D, is responsible for AE registration.
- *Goal and Policy Distribution Manager (GPM)* distributes goals and policies to corresponding AEs.
- *Life Monitor (LMO)* observes the liveness of AEs by listening to heartbeat messages from AEs. LMO regularly updates the liveness of AEs to REG.
- *Capability Administrator (CPA)* maintains and provides data about capabilities and their functionalities and performances in available nodes.
- *Capability Monitor (CPM)* monitors capabilities and sends updates to CPA.
- *Capability Allocation Manager (CAM)* generates (re-) configuration plans for AEs to be instantiated in nodes. CAM fetches the capability requirements and retrieves the capabilities from CPA. A configuration plan defines in which node an AE should execute. Configuration plans are generated based on capability requirements and policies. In addition, CAM allocates capabilities to AEs. The allocation depends on the capability structure and optimization criteria which can be specified in the policies.
- *Deployment and Instantiation Manager (DIM*) executes the configuration plan. It creates AEs in the defined nodes and assigns the behavior specifications.
- *Mobility Manager (MOM)* supports an AE when it is moved and re-instantiated in a new node. The move can be related to failures or insufficient capability performances. MOM broadcasts messages to inform other AEs when an AE's functionality is suspended or is resumed. MOM also handles an AE's connections by getting input messages on behalf of an AE and forwarding them to such AE when it is already re-instantiated.
- *Ontology Repository (OntRep)* stores the goal and policy concepts as well as the related capability and service concepts.
- *Service Specification Repository (SpcRep)* stores the AE behavior specifications and the capability requirements.
- *Capability Repository (CapRep)* stores data about available nodes and their capabilities.

VI. Case Study

A music video streaming system is presented with the intention to demonstrate the *Strategist* and *Judge* solution in the proposed autonomic framework. The system is constituted by the AEs as defined in Section V. The Primary Service AEs are *Streaming Manager (STM)* and *Streaming Client (STC)*. An STM, executing on a *media streaming server (MS)*, streams the music video files to STCs. An STC is associated with an SLA class, which defines *required streaming throughput*, *price for the service* and *service provider penalties* if the agreed QoS cannot be met. Two SLA classes are applied: *premium (P)* and *ordinary (O)*. An STC is denoted by its SLA class as STC_P or STC_O. Each SLA class has different required throughput (X); the STC_P required throughput (X_P) can be 1Mbps or 600Kbps for high-resolution and degraded fair-resolution videos, while the STC_O required throughput (X_O) is 500Kbps for low-resolution videos. Prices and penalties will be defined later.

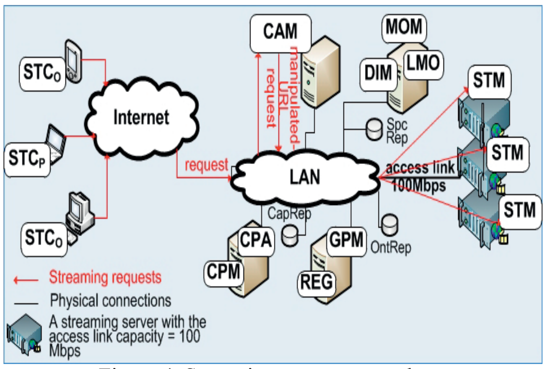


Figure 4. Streaming system example.

Figure 4 illustrates the streaming system. In this case study, CAM will accept the streaming requests on behalf of STMs. CAM will decide which an STM can serve the requests, or CAM may put them in waiting queues. CAM can also instantiate a new STM in an available MS that there is no executing STM.

The MS's required access link capacity ($C_{R,AL}$) is set to 100 Mbps. The number of STCs that can use the service at a time is limited by the MS access link capacity. When the required streaming throughput cannot be provided, a STC needs to wait until some connected requests have finished using the service. An STC_O can be disconnected, while an STC_P may have to degrade the video resolution. The service provider will pay penalties in case of waiting and disconnection of the STC. These penalty and price functions are given in Table II. A cost unit is the price paid by an ordinary client for one second streaming of the rate 500Kpbs. The price function for using the service is M(SLA_Class,X) (cost units/second). The penalty function for waiting is P_{WAIT}(SLA_Class) (cost units/second), and the penalty function for disconnection is P_{DISC}(SLA_Class) (cost units/connection).

Note that, the case study and all values, set in Table II, are same as our previous work [5, 6] in order to compare between the proposed and the previous model. The comparison results are in subsection VI.B.1.

Table II. The price and penalty functions.

	STC_O (X_O=500Kbps)	**STC_P (X_P=600Kbps)**	**STC_P (X_P=1Mbps)**
M(SLA_Class,X)/s	1	1.875	2
P_{WAIT}(SLA_Class)/s	5	10	10
P_{DISC}(SLA_Class)/ Connection	10	-	-

The complete set of actions A in this case study is:

$$A = \{a_D, a_B, a_N, a_I, a_R, a_T, a_M\} \tag{13}$$

A subset of A, Á, is defined as: $Á = A - \{a_M\}$. a_D is to disconnect the ordinary clients, a_B is to decrease the throughput of the premium clients, a_N is to instantiate a MS, a_I is to instantiate a new STM, a_R is to disconnect a MS, a_T is to terminate an STM and a_M is to move connected client sessions from an STM to another STM. These actions are selected by the *Strategist* of CAM. CAM executes a_N, a_I, a_R and a_T, while CAM suggests a_D, a_B and a_M to STMs.

In this case study, the considered capability is the MS access link. The required and inherent capability performance sets are denoted as $\hat{C}_R \equiv \{C_{R,AL}\}$ and $\hat{C}_I \equiv \{C_{I,AL}\}$, where $C_{R,AL}$ is the required access link capacity, and $C_{I,AL}$ is the available access link capacity. The inherent service performance set $\hat{S}_I$ 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 ordinary clients ($N_{Disc,O}$), the number of MS (N_{Node}), the service time and waiting time of premium and ordinary clients ($T_{Serv,P}$, $T_{Serv,O}$, $T_{Wait,P}$, $T_{Wait,O}$). These values as well as the inherent service income (I_I) are observed per a monitoring interval Δ. The service income is defined as:

$$\begin{aligned} I_I = {} & M(STC_O,X_O)*T_{Serv,O} + \\ & M(STC_P,X_P)*T_{Serv,P} - \\ & P_{WAIT}(STC_O)*T_{Wait,O} - \\ & P_{WAIT}(STC_P)*T_{Wait,P} - \\ & P_{DISC}(STC_O)*N_{Disc,O} - \\ & P_{Ser}*N_{Node}*\Delta \end{aligned} \tag{14}$$

where P_{Ser} is the cost function for adding a new MS which is 150 units/second per node, while M(SLA_Class,X), P_{WAIT}(SLA_Class) and

P_{DISC}(SLA_Class) are as already defined in Table II.

A. RM and LM Specification

In this case study, CAM plays an important role. Its RM specification is defined as follows:

$$R_{CAM} \equiv \{ \Theta_{CAM}, \Phi, \mathcal{P}_{CAM}, \xi_{CAM} \} \quad (15)$$

$\mathcal{P}_{CAM}$ consists of five policies (p_1-p_5) as presented in Appendix. A policy defines some actions in the set A in (13).

The LM specification of CAM is defined as follows:

$$L_{CAM} \equiv \{ \Omega_{CAM}, \Lambda, \Psi_{CAM}, \zeta_{CAM} \} \quad (16)$$

$$\Omega_{CAM} \equiv \{ g_1, g_2\} \quad (17)$$

$$g_1 \equiv (d_1: I_R > 0, \omega_1: 0.8) \quad (18)$$

$$g_2 \equiv (d_2: T_{Wait} < \Delta, \omega_2: 0.2) \quad (19)$$

where I_R is the required service income, and T_{Wait} is the sum of the waiting time of premium and ordinary clients. These goals are set in order to gain high income and to avoid high waiting time. The policy p_1-p_5 can be used when the required service income is not met, while the policy p_1-p_3 are used when the waiting time is higher than expected.

B. Experiments and Results

Two set of experiments are presented. In 0 Á is used for the comparison between the proposed and a previous model. In 0 different action sets A and Á are used to study the proposed model. The accumulated service income and the accumulated waiting time results are illustrated in both experiment sets.

The request arrivals are modeled as a Poisson process with an arrival intensity parameter λ_{SLA_Class}. The duration of streaming connections d_{SLA_Class} is constant and is set to 10 minutes. The traffic per MS access link ρ is defined as:

$$\rho = ((\lambda_P{*}d_P{*}X_P) + (\lambda_O{*}d_O{*}X_O))/ (N_{Node}{*}C_{I,AL}) \quad (20)$$

The monitoring interval Δ is 1 minute. The STCs will stop waiting and there is no penalty for waiting after 10 minutes. The number of available MS = 3. Initially, only one STM is instantiated.

I. Comparison between the proposed and a previous model

Our previous work [5, 6] presented an adaptation mechanism executed by a Reasoning Machine, which uses policies and goal to manage the adaptable systems. In the previous model, a policy consists of constraints and actions, and it is not associated with any specific conditions. So all defined policies will be executed when the systems are entering a reasoning condition. There is only one reasoning condition defined, i.e., the number of waiting clients > 0. In addition, only one goal based on the service income is used. The action is then rewarded by *the goodness score (QoX)* that is calculated by the percentage of the increased or decreased service income.

In this section, three different cases of ρ are illustrated: *a) ρ<1*, *b) ρ=1* and *c) ρ=1.15 (ρ>1)*. The STC_P request arrivals intensity (λ_P) is set to 25%, 50% and 80% of the total arrival intensity.

For case *a) ρ<1*, both models have the same behaviors. If ρ>0.5, they used {a_N, a_I} to instantiate a MS and to instantiate a new STM, otherwise they just disconnected the ordinary clients or decreased the throughput of the premium clients. The accumulated service income and waiting time of both models are almost the same.

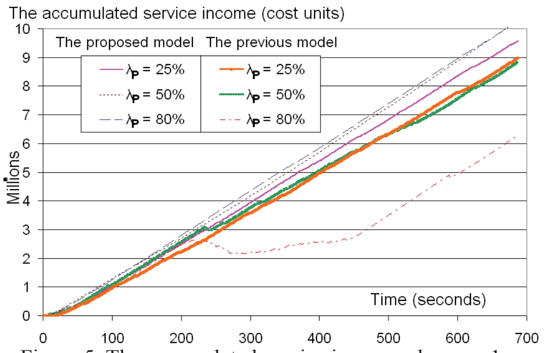


Figure 5. The accumulated service income when ρ = 1.

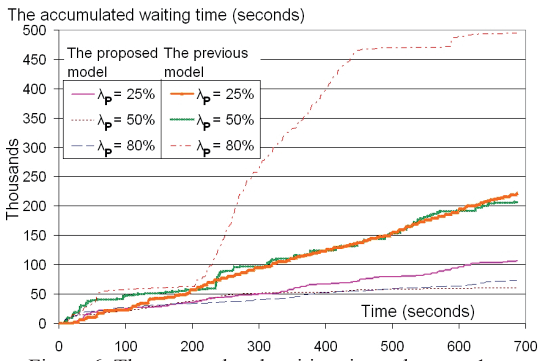


Figure 6. The accumulated waiting time when ρ = 1.

For case *b) ρ=1,* the proposed model produced higher accumulated service income and lower accumulated waiting time independent of λ_P, as depicted in Figure 5 and 6. This is because in the previous model $\{a_T, a_R\}$, which terminate an STM and disconnect a MS consecutively, was used when the number of waiting clients was little more than zero. So that, the number of MS was lower than proper required amount. It results in decreasing service income and increasing waiting time. In the proposed model, $\{a_T, a_R\}$ might be used only if the inherent service income <= 0; however, it does not happen when ρ = 1.

For case *c) ρ=1.15 (ρ>1)*, the proposed model also produced higher accumulated service income and lower accumulated waiting time. Figure 7 and 8 illustrates the accumulated service income and the accumulated waiting time for this case. The proposed model produced better results, because it took the actions to adapt the system more often than the previous model. When ρ>1, the service income could be less than 0 because of the waiting penalty. So that, in the proposed model the actions were applied both when the service income < 0 and when the waiting time > Δ, while the previous model the actions were applied only when the number of waiting clients > 0. However, when λ_P = 80% the traffic was too overloaded, and the accumulated service income was less than zero in both models.

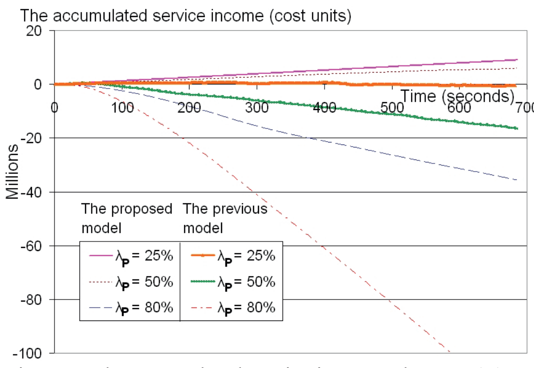


Figure 7. The accumulated service income when ρ = 1.15.

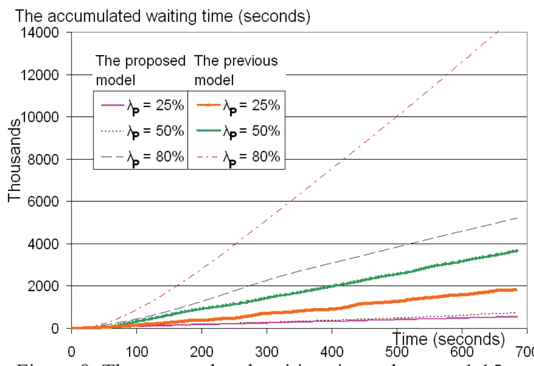


Figure 8. The accumulated waiting time when ρ = 1.15.

2. Comparison between different action sets

In this section, we compare three cases (I-III) of the new proposed model. In Case I the complete set of actions A is used, while in Case II the subset Á is used. For the last case, the complete set of actions A is also used, but there is no *Judge* component in the AEs so the actions are not rewarded. The traffics that were simulated for this scenario are relative to a function of time. The time with ρ at a fixed level, denoted as the *ρ period*, is set to 30 minutes. ρ varies from 0.2 to 1.2. λ_P is set to 50% of the total arrival intensity.

Figure 9 and 10 shows the accumulated service income and the accumulated waiting time of three cases. The brown line in these figures shows the variation of ρ. In Case I, the system learned that $\{a_M, a_T$ and $a_R\}$, which move connected STC sessions, terminate an STM and disconnect a MS consecutively, is efficient to adapt the system when ρ drops and then the required service income is not met. As a result, Case I could produce the highest accumulated service income and the lowest accumulated waiting time. For the last case, the actions were selected randomly and they were not appropriate to the states of unwanted service income and the waiting time. So, the accumulated service income of Case III was the lowest, while the accumulated waiting time was the highest.

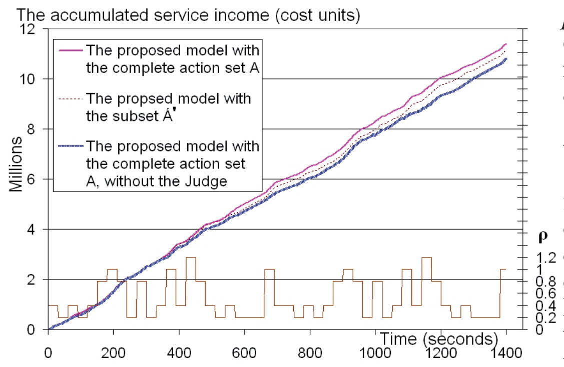


Figure 9. The accumulated service income for various ρ.

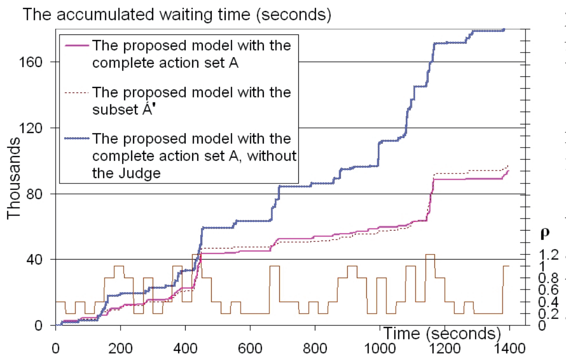


Figure 10. The accumulated waiting time for various ρ.

VII. Related Work

Existing service system frameworks that support run-time self-management and adaptation can be classified based on the way in which the management and adaptation functionalities are specified. The functionalities can be *statically* or *dynamically* specified. Some works propose to use templates [9] or adaptation classes [10] to statically specify these functionalities. However, the static approach lacks flexibility. All the possible adaptation must be known a beginning, and if new adaptations are required, the systems must be re-complied. Our work expresses the service management functionality for the adaptable service systems in the form of the EFSM, RM and LM specification, to be dynamically modified, added and removed at run-time. When using the EFSM specification, an *update* of changes is done by deployment of the whole specification. When using the RM and LM specification, only incremental changes of the *policies* and *goals* are deployed. However, the complete policy and goal based functionality need to be validated off-line before the deployment of the incremental changes.

There are several works that use the policies to specify the adaptation, such as [11], [5, 6], [12-15]. Accord [11] is a framework that can formulate autonomic applications as dynamic composition of AEs, with the use of policies to describe the adaptation of functional behaviors of AEs and interactions between them. However, our approach and the rest go beyond the use of policy for the specification by adding mechanisms to adapt policies or the way of using policies. Such policy adaptation can be grouped into three categories: 1) changing the policy parameters, considered in [5, 6, 12, 13]; 2) enabling/disabling a policy, found in [5, 6, 12]; 3) using techniques to select the most suitable policy and action; for instance, rewarding policies and their actions, presented in [14, 15].

Our approach is an instance of the first as well as the third category as [14, 15]. Tesauro et al. [14] presented a hybrid reinforcement technique used for resource allocation in multi-application data centers. This technique is to select optimal policies that can maximize rewards. Mesnier et al. [15] used decision trees to select accurate policies in storage systems. These policy adaptation techniques have only been applied to a single element, while our approach is potentially used in multi-autonomic elements.

VIII. Conclusions

This paper proposed an autonomic framework for adaptable service systems. The framework solution consists of *Goal-based Policy Ontology* and *Autonomic Element (AE) Model*. The Ontology defines common concepts of *goal*, *policy* and *inherent state*. AEs are generic component that can be used to realize any functionality. An AE is constituted by *Main Function*, *Strategist*, *Judge* and *Communicator* modules. The functionality of an AE is realized by two Extended Finite State Machines (EFSM), one Reasoning Machine (RM) and one Learning Machine (LM). EFSM behavior, as well as goals and policies can be modified flexibly during run-time. For attaining a specific functionality, specific EFSM, RM and LM functionality must be defined. In this paper, specific AEs handling service management functionality is proposed.

A case study is presented with focus on the Capability Allocation Manager (CAM). The experimental results show that the proposed model can produce higher service income and less waiting time than a previous model. In the proposed model, the actions are used appropriately under the associated goals and required goal measures. Moreover, it is possible to apply several goals, which each are weighed differently, depending on its importance. New actions can be added, and when there are more actions the system may reach the goals quicker.

References

[1] P. Thongtra and F. A. Aagesen. Capability Ontology in Adaptable Service System Framework. In Proc. of 5th Int. Multi-Conference on Computing in the Global Information Technology, Spain, Sep 2010.

[2] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. IEEE Computer Society, January 2003, pp. 41-47.

[3] S. White, J. Hanson, I. Whalley, D. Chess, and J. Kephart. An architectural approach to autonomic computing. In Proc. of 1st IEEE Int. Conf. on autonomic computing, New York, May 2004, pp. 2–9.

[4] P. Thongtra and F. A. Aagesen. An Adaptable Capability Monitoring System. In Proc. of 6th Int. Conference on Networking and Services (ICNS 2010), Mexico, March, 2010.

[5] P. Supadulchai and F. A. Aagesen. Policy-based Adaptable Service Systems Architecture. In Proc. of 21st IEEE Int. Conf. on Advanced Information Networking and Applications (AINA'07), Canada, 2007.

[6] P. Supadulchai, F. A. Aagesen and P. Thongtra. Towards Policy-Supported Adaptable Service Systems. EUNICE 13th EUNICE Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services. Lecture Notes in Computer Science (LCNS) 4606, pp 128-140.

[7] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge Engineering: princicples and methods. Data & Knowledge Engineering, vol. 25, pp. 161-197, 1998.

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

[9] F. Berman, R. Wolski, H. Casanova, et al. Adaptive computing on the grid using AppLeS. In IEEE Trans. Parallel Distrib. Syst., vol. 14, no. 4, pp. 369–382, Apr. 2003.

[10] P. Boinot, R. Marlet, J. Noy´e, G. Muller, and C. Cosell. A declarative approach for designing and developing adaptive components. In Proc. of the 15th IEEE Int. Conf. on Automated Software Engineering, 2000.

[11] H. Liu and M. Parashar. Accord: a programming framework for autonomic applications. In IEEE Trans. on System, Man, and Cybernetics, vol. 36, pp. 341–352, 2006.

[12] L. Lymberopoulos, E.C. Lupu and M.S. Sloman. An Adaptive Policy-Based Framework for Network Services Management. In Journal of Networks and Systems Management, vol. 11, pp. 277–303, 2003.

[13] K. Yoshihara, M. Isomura, and H. Horiuchi. Distributed Policy-based Management Enabling Policy Adaptation on Monitoring using Active Network Technology. In Proc. of 12th IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management, France, Oct 2001.

[14] G. Tesauro, R. Das, N.K. Jong, and M.N. Bennani. A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation. In Proc. of 3rd IEEE Int. Conf. on Autonomic Computing (ICAC'06), Ireland, Jun 2006, pp. 65–73.

[15] M. Mesnier, E. Thereska, D. Ellard, G.R. Ganger, G.R., and M. Seltzer. File classification in self-* storage systems. In Proc. of Int. Conf. on Autonomic Computing (ICAC-04), pp. 44–51.

[16] W3C, "OWL Web Ontology Language Overview," 2004 Available at: http://www.w3.org/TR/owl-features/

[17] V. Wuwonse and M. Yoshikawa. Towards a language for metadata schemas for interoperability. In Proc. of 4th Int. Conf. on Dublin Core and Metadata Applications, China, 2004.

Appendix

The appendix includes the policy specifications and a list of mathematical expressions found in this paper.

A: Policy Specifications

The policies as well as goals are expressed in OWL (Web Ontology Language) [16] and OWL/XDD (XML Declarative Description) [17], where the variables can be integrated with ordinary OWL elements. The variables are prefixed with the $ sign. In this paper, the policy is written in the form:

Conditions: Expressions_for_conditions,
Constraints: Expression_for_constraints,
Actions: {Action_ID},
Operation cost: Expression_for_operation_cost

Five policies (p_1-p_5) used in the case study are listed in Table III. The conditions can be the inherent service income $\$I_I <= 0$ and the waiting time $\$T_{Wait} >= \Delta$, where $\$T_{Wait} = \$T_{Wait,P} + \$T_{Wait,O}$.

Table III. The Policy Set

p_1	**Conditions:** $\$I_I <= 0$ or $\$T_{Wait} >= \Delta$, **Constraints:** $P_{WAIT}(STC_O) < P_{WAIT}(STC_P)$, **Actions:** $\{a_D\}$, **Operation Cost:** $P_{DISC}(STC_O)$ This policy can be read as: a_D should be used to disconnect a list of STC_O when $P_{WAIT}(STC_O) < P_{WAIT}(STC_P)$, and the number of STC_O being disconnected is calculated from $X_{P,1Mbps} * \$N_{Wait,P} / X_O$. a_D costs $P_{DISC}(STC_O)$ units.
p_2	**Conditions:** $\$I_I <= 0$ or $\$T_{Wait} >= \Delta$, **Constraints:** $P_{WAIT}(STC_O) > M(STC_P,X_{P,1Mbps})- M(STC_P,X_{P,600Kbps})$, **Actions:** $\{a_B\}$, **Operation Cost:** $M(STC_P,X_{P,1Mbps}) - M(STC_P,X_{P,600Kbps})$ This policy can be read as: a_B should be used to decrease the throughput of a list of STC_P when $P_{WAIT}(STC_O) > M(STC_P,X_{P,1Mbps}) - M(STC_P,X_{P,600Kbps})$, and the number of STC_P to decrease the throughput is calculated from $X_O * \$N_{Wait,O} / (X_{P,1Mbps} - X_{P,600Kbps})$. a_B costs $M(STC_P,X_{P,1Mbps}) - M(STC_P,X_{P,600Kbps})$.
p_3	**Conditions:** $\$I_I <= 0$ or $\$T_{Wait} >= \Delta$, **Constraints:** $(X_{P,\ 1Mbps} * \$N_{Wait,P} + X_O * \$N_{Wait,O}) / C_{R,AL} > 0.1$, **Actions:** $\{a_N, a_I\}$, **Operation Cost:** $P_{Ser} * \Delta$ This policy can be read as: a_N and a_I should be used to instantiate a MS and to instantiate a new STM consecutively, when $(X_{P,\ 1Mbps} * \$N_{Wait,P} + X_O * \$N_{Wait,O}) / C_{R,AL} > 0.1$. These actions $\{a_N, a_I\}$ cost $P_{Ser} * \Delta$.
p_4	**Conditions:** $\$I_I <= 0$, **Constraints:** $(X_{P,\ 1Mbps} * \$N_{Wait,P} + X_O * \$N_{Wait,O}) / C_{R,AL} < 0.1$, **Actions:** $\{a_T, a_R\}$, **Operation Cost:** $P_{DISC}(STC_O) + P_{WAIT}(STC_P) – P_{Ser} * \Delta$ This policy can be read as: a_T and a_R should be used to terminate an STM and to disconnect a MS consecutively, when $(X_{P,\ 1Mbps} * \$N_{Wait,P} + X_O * \$N_{Wait,O}) / C_{R,AL} < 0.1$. These actions $\{a_T, a_R\}$ cost $P_{DISC}(STC_O) + P_{WAIT}(STC_P) – P_{Ser} * \Delta$.
p_5	**Conditions:** $\$I_I <= 0$, **Constraints:** $(X_{P,\ 1Mbps} * \$N_{Wait,P} + X_O * \$N_{Wait,O}) / C_{R,AL} < 0.1$, **Actions:** $\{a_M, a_T, a_R\}$, **Operation Cost:** $- P_{Ser} * \Delta$ This policy can be read as: a_M, a_T and a_R should be used to move connected STC sessions, to terminate an STM and to disconnect a MS consecutively, when $(X_{P,\ 1Mbps} * \$N_{Wait,P} + X_O * \$N_{Wait,O}) / C_{R,AL} < 0.1$. These actions $\{a_M, a_T, a_R\}$ make profit = $P_{Ser} * \Delta$.

B: Mathematical Expressions

Table IV lists all mathematical expressions. This table also expresses the relations to others expressions (Rel. to exp.) as well as the references to table (Ref. to tab.), where the notification used in the expression are defined.

Table IV. Mathmatical Expressions.

No.	Mathematical expressions	Rel. to exp.	Ref. to tab.
1	$E \equiv \{ S_M, S_I, S_S, V, M, O, Q, F_S, F_O, F_V \}$	-	-
2	$R \equiv \{ \Theta, \Phi, \mathcal{P}, \xi \}$	-	-
3	$\xi \equiv (\overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I, \hat{C}_{A,n}; n=[1, N])$	2	I
4	$\Pi \equiv \{ p_i \}$	2	-
5	$p_i \equiv (\Sigma_i, X_i, A_i)$	4	-
6	$\Sigma_i \equiv Expression(\overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I)$	5	I
7	$X_i \equiv Expression(\overline{S_R}, \hat{S}_R, \overline{C_R}, \hat{C}_R, I_R, \overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I, \hat{C}_{A,n}; n=[1, N], \Gamma)$	5	I
8	$L \equiv \{ \Omega, \Lambda, \Psi, \zeta \}$	-	-
9	$\zeta \equiv (\overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I)$	8	I
10	$\Omega \equiv \{ g_k \}$	8	-
11	$g_k \equiv (d_k, \omega_k)$	10	-
12	$reward(a_i,i_{k,t-1},d_k) = (\Delta(i_{k,t},i_{k,t-1})/\Delta(d_k,i_{k,t-1}))*\omega_k -cost(a_i)$	8	

13	$A = \{a_D, a_B, a_N, a_I, a_R, a_T, a_M\}$	5	-
14	$I_I = M(STC_O,X_O)*T_{Serv,O} + M(STC_P,X_P)*T_{Serv,P} - P_{WAIT}(STC_O)*T_{Wait,O} - P_{WAIT}(STC_P)*T_{Wait,P} - P_{DISC}(STC_O)*N_{Disc,O} - P_{Ser}*N_{Node}*\Delta$	3, 6, 7, 9	II
15	$R_{CAM} \equiv \{ \Theta_{CAM}, \Phi, \Pi_{CAM}, \xi_{CAM} \}$	2	-
16	$L_{CAM} \equiv \{ \Omega_{CAM}, \Lambda, \Psi_{CAM}, \zeta_{CAM} \}$	8	-
17	$\Omega_{CAM} \equiv \{ g_1, g_2\}$	10, 16	-
18	$g_1 \equiv (d_1: I_R > 0, \omega_1: 0.8)$	11, 17	-
19	$g_2 \equiv (d_2: T_{Wait} < \Delta, \omega_2: 0.2)$	11, 17	-
20	$\rho = ((\lambda_P*d_P*X_P) + (\lambda_O*d_O*X_O))/(N_{Node}*C_{I,AL})$	-	-

Paper G: On Capability-related Adaptation in Networked Service Systems

Finn Arve Aagesen and Patcharee Thongtra

Int. Journal of Computer Networks & Communications (IJCNC), vol. 4, no. 4, July 2012.

On Capability-related Adaptation in Networked Service Systems

Finn Arve Aagesen and Patcharee Thongtra

Department of Telematics, NTNU, Trondheim, Norway
{finnarve, patt}@item.ntnu.no

ABSTRACT

Adaptability is a property related to engineering as well as to the execution of networked service systems. This publication considers issues of adaptability both within a general and a scoped view. The general view considers issues of adaptation at two levels: 1) System of entities, functions and adaptability types, and 2) Architectures supporting adaptability. Adaptability types defined are capability-related, functionality-related and context-related adaptation. The scoped view of the publication is focusing on capability-related adaptation. A dynamic goal-based policy ontology is presented. The adaptation functionality is realized by the combination of extended finite state machines, reasoning machines and learning mechanisms. An example case demonstrating the use of a dynamic goal-based policy is presented.

KEYWORDS

Adaptable service systems, Adaptability types, Adaptability architecture, Capability-based adaptation, Goal-based policy ontology, Policy-based adaptation

1. Introduction

Networked service systems are considered. *Services* are realized by *service components* which by their inter-working constitutes a *service system*. 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 a system for the *specification*, *management* and *execution* of *service systems*.

Adaptability can generally be defined as *the ability of a system to fit to changed circumstances*. Adaptability is generally realised by some closed feed-back loop. Fitting behavior can be of various types and can take place several levels. It is tendency, however, to denote many aspects of changes or "fitting behavior" as adaptation, without setting requirements to which changes that "qualifies" with *changed circumstances*. An *autonomic service system* is a specialization of an adaptable service system. Autonomic systems have ability to manage themselves and to adapt dynamically to changes in accordance with given objectives [1,2]. An autonomic system is constituted by distributed components denoted as *autonomic elements*.

The contribution of this publication is a general concept framework for adaptable systems as well as a scoped framework for capability-related adaptation. Adaptable service systems are accordingly considered within a *general view* and within a *scoped view*. Within the *general view* various adaptability issues of networked service systems are considered at two levels. *Entities*, *functions* and *adaptability types* are considered at Level 1, *Architectures* supporting adaptability at Level 2. The adaptability *types* defined are *capability-related*, *functionality-related* and *context-related* adaptation. Within the *scoped view*, *capability-related* adaptation is focused. A goal-based policy ontology for capability-related adaptation is presented. The experience background for this publication is work with the TAPAS *architecture* and *platform* [3-5]. Another important reference is the FOCALE architecture for autonomic networking [6]. Even if parts of this publication are inspired by TAPAS, the intention is to be rather generic.

The rest of the publication is organized as follows. Section 2 considers related works. In Section 3 important performance concepts related to capabilities and services are defined. Section 4-5 presents the Level 1 and 2 issues as defined above. Section 4 presents entities and adaptability functions, while Section 5 presents adaptability types. Section 6 presents adaptability supporting architectures. In Section 7-8 capability-related adaptation is handled. Section 7 defines a goal-based policy ontology, and Section 8 presents an example case demonstrating the use of this ontology. Summary and conclusions are presented in Section 9.

2. Related Works

In Section 1 and the Sections 3-7 some references to related works [1-21] are presented. In this Section additional references [22-28] are provided. Existing service system frameworks that support run-time self-management and adaptation can be classified according to how the management and adaptation functionalities are specified. Some works propose to use templates [22] or adaptation classes [23] for specification. However, such approach lacks flexibility. All possible adaptation cases must be known, and new adaptation cases require re-compilation. The architecture presented in this publication specifies the adaptability functionality based on Extended Finite State Machines, Reasoning Machines and Learning Mechanisms, to be dynamically modified during run-time. For Extended Finite State Machine specifications, an update of changes is done by deployment of the whole specification. For Reasoning Machine and Learning Mechanism, only incremental changes of policies and goals are deployed. The complete policy and goal based functionality, however, need to be validated before deployment of the incremental changes.

There are several works that use policies to specify the adaptation, such as [3, 4], [24], [25-28]. In [24] a framework that defines autonomic applications as dynamic composition of autonomic elements is described. Our approach as well as the approaches described in [3, 4, 25-28] go beyond by adding mechanisms to adapt policies or the way of using policies. Such policy adaptation can be grouped into three categories: 1) changing the policy parameters, considered in [3, 4, 25, 26]; 2) enabling/disabling a policy, found in [3, 4, 25]; 3) using techniques to select the most suitable policy and action; for instance, rewarding policies and their actions, presented in [27, 28]. Our approach is of category 1 and 3. Tesauro et al. [27] presented a hybrid reinforcement technique used for resource allocation in multi-application data centers.

This technique is to select optimal policies that can maximize rewards. Mesnier et al. [28] used decision trees to select accurate policies in storage systems.

3. Capability and Service Performance Concepts

A capability is here defined as an *inherent property* of a node used as a basis to implement services. A service component may need capabilities to be allocated before deployment and instantiation. Capability types are classified as *resources*, *functions* and *data*. Resource examples are CPU, memory, transmission links, sensors and batteries. Within network management the concept managed objects is used. Managed objects such as MIB (Management Information Base) objects in SNMP [7] and CIM (Common Information Model) objects in WBEM (Web-based Enterprise Management) [8] are considered as capabilities.

Capability parameter describes the characteristics of a capability type and can be classified as functionality, performance and inference parameters [9]. *Functionality* parameters define functionality features, *performance* parameters define performance measures and *inference* parameters define logical relations to other capability types. Capability *performance parameters* can further be classified as: *capacity* parameters, *state* parameters and *QoS* parameters. Capability capacity parameters examples are transmission channel capacity, the number of streaming connections and CPU processing speed. Capability state parameters examples are number of connections, and the number that is waiting. Capability QoS parameter examples are transfer time, throughput, utilization, availability and recovery time after errors. The services provided to the service user can in the same way as capabilities be described by *functional* and *performance* parameters. *Service performance parameters* are further classified as state and QoS parameters. The capability and service performance provided at an observed time instance or during an observation time interval are denoted as *inherent* capability and service performance.

Service level agreements (SLA) are agreements between the service users and the service provider. The agreement can contain elements such as: required service functionalities and performance, payment for the service when the agreed performance is offered and penalty in case of reduced performance. The service can be differentiated in various *QoS classes*. This QoS class will be reflected in the SLA. A service component can generally handle user services related to several QoS classes.

4. Entities and Functions

4.1. Entities

Entities related to the life-cycle of an adaptable service system are illustrated in Figure 1. The *service framework* is constituted by the *Primary Service System* itself, the *Service Creation System*, the *Service Repository*, and the *Network and Service Management System*. The *environment* is constituted by the administrative *Service Provider* and the *Service User*. The Primary Service System provides services to the Service Users, while the Network and Service Management System provides management services to the

Primary Service System. Both the Primary Service System and the Network and Service Management System are service systems according to the definitions in Section 1. The boundary between these systems highly depends on the nature of the primary service, the nature of the service management functionality, and how the various functions are realized in software components. Some management functionalities can be integrated in the software components executing the primary service functionality. In the figure this is denoted as delegated management.

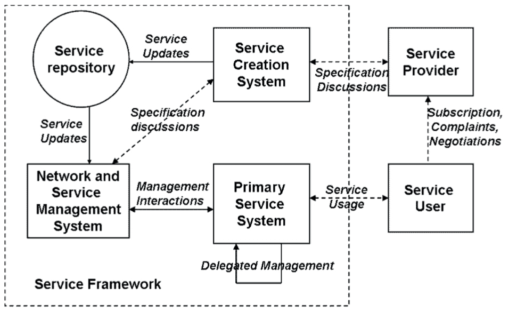


Figure 1. Service life-cycle Entities

4.2. Functions

The functionalities of the service framework entities are here grouped in the *functionality groups*: Service Creation, Node Configuration, Service Configuration, Service Provisioning, and Monitoring and Diagnosis as illustrated in Figure 2. These groups are not absolute and the grouping can be done in several ways. The functions in the functionality groups are rather not necessarily single functions and can in some cases be considered as functionality sub-groups. With reference to Figure 1, Service Creation is done by the Service Creation System. Service Creation comprises the specification, integration and validation of the service system components. Node Configuration, Service Configuration, Service Provisioning, and Monitoring and Diagnosis are done *in cooperation* between the Network and Service Management System and the Primary Service System.

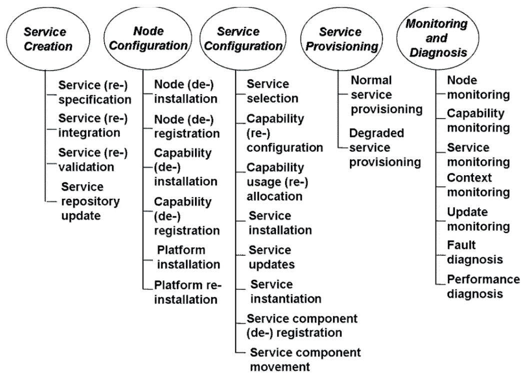


Figure 2. Service Life-cycle Functionalities

Concerning the sub-functionalities of *Node Configuration*, *installation* is the provision of the physical existence of the component, while *registration* is the logical registration of nodes and capability instances. The physical installation is done by humans, while the logical registration can be done by a service system. But this functionality will normally require cooperation with a *monitoring function* that can detect the physical presence of the component. Platform re-installation can generally be the installation done after a failure of a node, but can also be the installation of a new version of the platform software.

Concerning *the sub-functionalities of Service Configuration*, *Service selection* is primarily a function used in cases where change of context needs a new service. *Capability configuration* comprises *Capability parameter configuration* as well as *Node selection*. *Capability parameter configuration* is the validation and settings of node capability parameter values according to a capability parameter configuration specification. *Node selection* is the selection of node with respect to the required capability functionality and performance defined for the service. *Capability usage allocation* determines the usage of allocated capabilities. *Service installation* is the deployment of the service components constituting a service system. *Service update* is here indicated as a different function. This is because an update does not need to comprise a new complete installation of all service components constituting a service system. *Instantiation* starts the execution of the installed or updated service components. *Service component (de-) registration* is the (de-)registration of the service component instance in a system that provides an overview of the instantiated service component instances. *Capability re-configuration* can in general initiate the movement of service components. *Service component movement* is different from installation in general as it includes the movement of an instance of a service component with present states and local variable values

Service Provisioning comprises *Normal service provisioning* as well as *Degraded service provisioning* as realised by the Primary Service System as illustrated in Figure 1. Degraded here refers to degraded service and capability performance that require adaptation actions. With respect to Figure 2, adaptation actions can be Capability usage re-allocation only, but in more serious cases it can comprise *Capability re-configuration*, *Capability usage re-allocation*, *Service component movement*, *Service component instantiation*, as well as *Service component de-registration* and *Service component registration*.

Concerning the sub-functionalities of *Monitoring and Diagnosis*, *Node monitoring* monitors the existence and liveness of nodes. *Capability monitoring* monitors the existence of capabilities types and parameter values. *Service monitoring* similarly monitors the existence, liveness, functionality and performance parameters of service components. *Update monitoring* is the monitoring for the existence of service system software updates in the service repository. *Fault diagnosis* detects failures related to nodes, capabilities and service components, while *Performance diagnosis* detects mismatch between required and inherent service and capability performance.

Context is here related to the *adaptability terrain*, i.e. the environment in which the adaptable system operates. The terrain can be classified as *physical* or *logical*. *Physical terrain context* is defined by physical positions as well as the state of the terrain. Physical positions can be absolute position of the node executing the adaptable service system as well as positions relatively to other nodes and objects. State measures can be terrain type, temperature, concentration of gases, friction, fluidity, etc. A *logical terrain* is defined by *logical positions* and application layer *associations* between service components. Physical terrain examples can be streets, buildings on fire, collapsed buildings, water and nature. One adaptable system scenario is a robot snake [10] operating in a house on fire changing the movement type according to the terrain type as well as to blocked ways. Service examples are measurements, searching humans, video recording and the spraying of water. As nodes operating in a physical terrain also can have wire-less communication, they will both operate in a physical and logical terrain. One example service system is a robot soccer play where the various players in the team has roles and interact logically in addition to the behaviour defined from the monitoring made by physical sensors [11]. Context can also be defined to include capabilities, such as memory, CPU, battery, bandwidth as well as user preferences and profile [12]. User preferences and profile are here considered as data and are not visible in the functionality models presented.

5. Adaptability Type and Functions

Adaptability types are here classified as Capability-related, Functionality-related and Context-related adaptation. *Capability-related* adaptation is here defined as the ability to adapt because of shortage of capabilities with appropriate logical functionality or overload or failure. *Functionality-related* adaptation is the ability to adapt to new functionality requirements, and *Context-related* adaptation is the ability to adapt to context changes as defined in Section 4.

As capabilities are the basic fundament for the implementation of service systems, a needed property for any adaptability type is *to be aware of node and capabilities* and the ability to *configure service systems* according to the present availability of nodes and capabilities. This is denoted as *basic capability awareness.* The needed functionality is illustrated in Figure 3. In this case there is assumingly no failures and overload. Installation of new versions of the platform software is not part of the functionality considered in the architecture models presented in this publication. The functionalities needed for the various adaptability types are defined as follows:

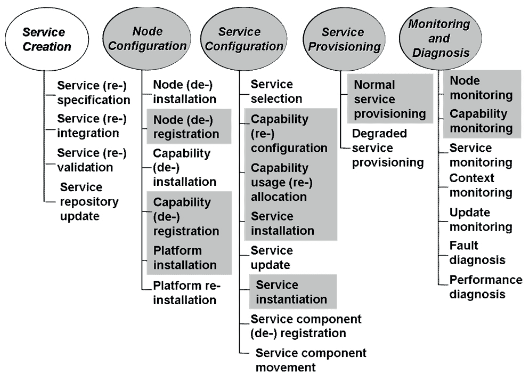


Figure 3. Basic Capability Awareness

1) *Capability-related adaptation* needs functionality as illustrated in Figure 4. The functionality needed for basic capability awareness is illustrated in grey. The added needed functionality needed is illustrated in blue. In addition to the Basic Capability Awareness Platform (re-) installation, Service component (de-) registration, Service component movement, Degraded service provisioning, Service monitoring, Fault diagnosis and Performance diagnosis is needed.

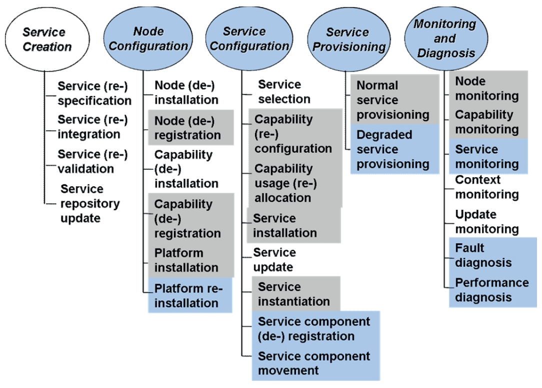


Figure 4. Capability-Related Adaptation

2) *Functionality-related adaptation*: In addition to the functionalities of Capability-related adaptation, the following functionality is needed: Service (re-) specification, Service (re-) integration, Service (re-) validaton, Service repository update, Update monotoring and Service updating. Fault and failure diagnosis are included for covering the cases where faults or performance leads to redesign of the service system software. Degraded service provisioning covers cases where new software needs capability re-configuration because of reduced performance. Service creation functionalities must have involvement by humans. The functionalities that can be automated are Service repository update and Service update [13].

3) *Context-related adaptation*: In addition to Basic capability awareness the following functionality is needed: Service Monitoring, Context monitoring and Service Selection. Context-related adaptation is related to terrain as defined in Section 4. Context defined by physical position is used in a wide variety of commercial user services related to ticket sales, visiting touristic places, restaurant services etc. [12]. Context-related adaptation is to some extension pre-programmed. Physical terrain adaptation is programmed mathematically. Logical adaptation is realized by client applications, i.e. client service components that flexibly are able to interwork with new server service components. Mostly user interaction is also needed to select among the available applications.

6. Adaptability Supporting Architectures

6.1. General

The service framework is constituted by two architectures: computing architecture and service functionality architecture. This is the same architecture structure as applied in TINA [14]. The computing architecture has concepts for the specification of service system behaviour. The architecture that describes the structure of services functionalities is denoted as service functionality architecture. An execution platform must both i) support the concepts of the computing architecture and ii) provide management support for the service systems as illustrated in Figure 5. The management functionality which is a part of the service functionality architecture comprises both network and service management functionality.

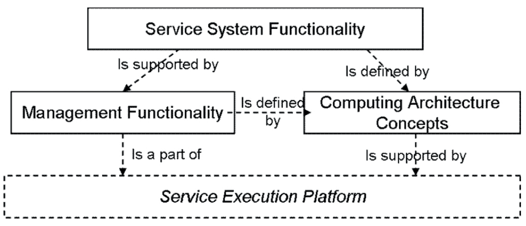


Figure 5. Functionality, Architecture and Platform

6.2. Computing Architecture

6.2.1. Service Layer and Physical Layer

A computing architecture based on two abstraction layers is illustrated in Figure 6. These two layers are denoted as service layer and physical layer. This is a generalization of the three-layer TAPAS model [3]. Several architecture models with a variable number of layers have been proposed. In [15], a five-layer model is presented.

The service layer defines the service constituted by service components. Leaf service components are realized by Actors. Actors are executed as operating system software components. The service system is specified by EFSMs (Extended Finite State Machines), goals and policies. Actors are EFSM interpreting mechanisms. The service components are implemented by a combination of Actors, Reasoning Machine (RM) and Learning Mechanisms (LM). EFSMs can activate RM and LM, and the EFSM can use the RM to select appropriate actions in situation when strategic decisions are needed. RM and LM models will be described in Section 7.

6.2.2. Service Component Features

The service components have features in addition to the service behavior execution. The following features must be supported: i) Renewal of Service Component EFSM behavior specification ii) Renewal of Policy and Goal specifications, iii) Movement of Service component while preserving state, variables and messages, and iv) Management of Service component EFSM states.

These features need support of management functionality. During runtime new versions EFSM functionality, policy and goal specifications can be downloaded and instantiated. Service components can be moved by instantiating a new Actor in a new node. Some of the EFSM states are classified as stable states. A stable state is a state where the functionality of a service component can move safely and be re-instantiated. Re-instantiation includes the restoration of EFSM state, EFSM local variables, and queued messages.

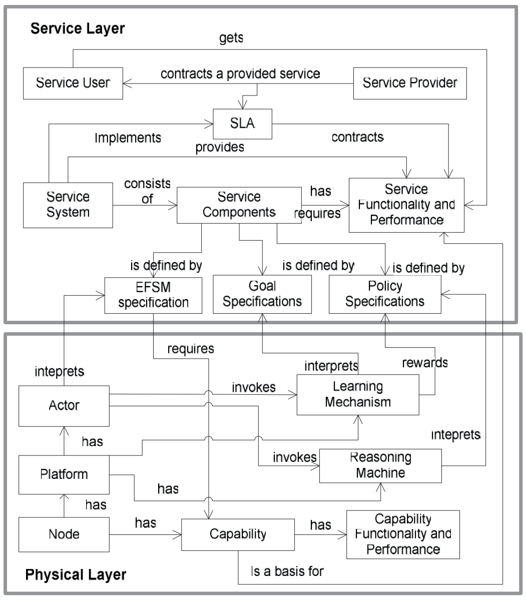


Figure 6. Computing Architecture

Actors can manage its EFSM states and local variables based on received EFSM input messages as well as responses from the Reasoning Machine. The generic Actor states are: {Initial, Normal, Degraded, Moving, Idle, Terminated}. A service component is instantiated in Initial state in a node where the required capability functionalities and performances are met. In Normal state services are provided with satisfactory performance. In Degraded state the performance is considered too low. Adaptation can be initiated to return to Normal state. In Moving state an Actor is moved and re-instantiated in a new node. In Idle state the execution of current EFSM specification is ended and allocated capabilities are released. From this state an Actor can be initiated as new service component in Initial State.

6.2.3. Computing Architecture and Adaptability Features

As previously stated, the adaptability functionality is realized by a combination of concepts and mechanisms of the computing architecture and supporting management functionality. The computing architecture has a service layer and a physical layer and has functionality and performance concepts both related to capability and service. This provides a basis for basic capability-related awareness, which is the basis for all adaptation types defined. The architecture further opens for service systems defined by flexible combination of EFSM-, goal-, and policy specifications combined with learning mechanisms. The two-layer architecture combined with the flexible Actor execution behaviour also makes adaptation during runtime possible. The model as defined, however, is open with respect to the goal and policy ontology applied and also to the realization of Reasoning machine and Learning mechanism. This basic model should make it possible to define and implement all adaptation types as defined in Section 5.

Concerning functionality-related adaptation, it is emphasized that is only the service configuration features of functionality-related adaptation that is feasible. Concerning context-related adaptation the model is open to any use of sensor capabilities. This model is however a generic model. In specific application cases, refinement and elaboration is needed.

6.3. Service Functionality Architecture

The service functionality architecture consists of primary service functionalities and the management functionality components as illustrated in Figure 7. The following five repositories are defined: Service specification repository (SpcRep), Capability type repository (CapRep), Inherent capability and service repository (InhRep), Context repository (ConRep) and Platform repository (PltRep). SpcRep stores the services behavior specifications, SLAs and required capability functionality and performance specifications. CapRep stores the capability type concepts. Several languages are used for capability type definition. Examples are SNMP MIB-objects defined by ASN.1 [7], WBEM CIM objects defined by XML [8] and NETCONF [16] objects based on YANG [17]. In TAPAS platform [5] capability ontology is represented by OWL [18] and OWL/XDD [19]. InhRep stores data about available nodes, capability instances and instantiated service components. This comprises the address of the Actor realizing the service component, service type reference, state of the Actor, and Capability and Service performance parameter values. ConRep stores predefined context states while PltRep stores programs needed to execute service functionality. With respect to the physical view of the computing architecture, PltRep comprises Actor, Reasoning machine and Learning mechanism software.

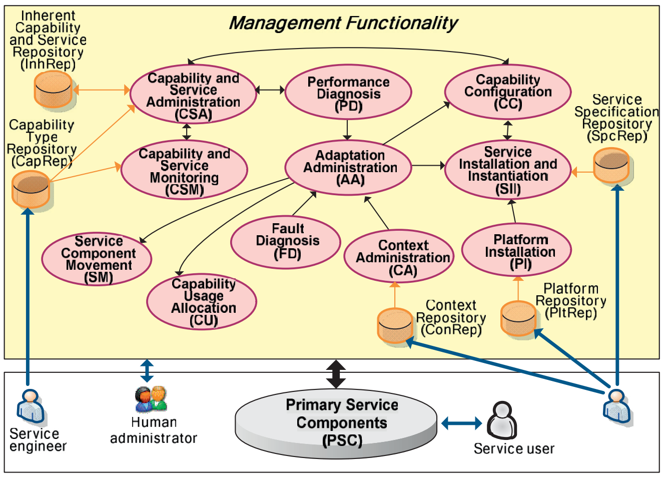

Management Functionality
Inherent Capability and Service Repository (InhRep)
Capability Type Repository (CapRep)
Capability and Service Administration (CSA)
Performance Diagnosis (PD)
Capability Configuration (CC)
Service Specification Repository (SpcRep)
Capability and Service Monitoring (CSM)
Adaptation Administration (AA)
Service Installation and Instantiation (SII)
Service Component Movement (SM)
Fault Diagnosis (FD)
Context Administration (CA)
Platform Installation (PI)
Capability Usage Allocation (CU)
Context Repository (ConRep)
Platform Repository (PltRep)
Service engineer
Human administrator
Primary Service Components (PSC)
Service user

Figure 7. Service Functionality Architecture

The functionality components as illustrated in Figure 7 implement functions for capability-related, functionality-related and context-related adaptation as defined in Section 5. The functions Capability configuration (CC), Capability usage allocation (CU), Service component movement (SM), Platform installation (PI), Fault diagnosis (FD) and Performance diagnosis (PD) correspond directly to functions defined in Section 4.

Capability and service administration (CSA) performs Node (de-) registration, Capability (de-) registration and Service component (de-)registration. A view of InhRep is also provided. *Capability and service monitoring* (CSM) performs Node, Capability and Service monitoring. The result of the monitoring is reported to CSA. *Capability configuration* (CC) generates configuration plans for service components. A configuration plan defines the node for deployment and instantiation. *Capability usage allocation* (CU) allocates capabilities in accordance with the present performance, SLAs and the optimization criteria chosen by the service provider. *Service Installation and instantiation* (SII) comprises deployment of EFSM, goal and policy specifications, as well as the execution of the configuration plan. *Service component movement* (SM) manages the ongoing sessions on behalf of a moving service component during movement. In TAPAS it forwards received messages after a service component is re-instantiated in a new location. SM will by broadcasting inform other service components of the “Moving” and “Normal” states of the moving component

Context Administrator (CA) monitors context and initiates configuration of context-dependent application software. Platform Installation (PI) is bootstrap functionality installing needed platform software when nodes are (re-) started. *Adaptation administrator* (AA) plans and administers re-configuration initiated by non-wanted events or states during the normal service system execution. This can be initiated by FD, PD, CA or by Human administrator. The re-configuration can result in CU only, or the combination of CC and CU, including SM and SII.

Primary service components basically implement the service provisioning as illustrated in Figure 2. There is, however, not always a clear boundary between primary service functionality and management functionality. Most primary service systems need capabilities and functionality components such as *PD*, *CU* and *CC*. Such functionalities can often be designed as part of the primary service system.

7. A Goal-based Policy Ontology for Capability-related Adaptation

An ontology is a formal and explicit specification of a *shared* conceptualization [20], containing both object types and functions operating on instances of object types. We can define independent concepts and relational concepts. Logic concepts can be defined by mathematical logics, e.g., if-then-else or by rules [9].

7.1. A Static Model based-on Reasoning Machine

Figure 8 presents a goal-based policy ontology. At the top level we have *goal*, *policy* and *inherent state*. This model is denoted as *static* because there is no feed-back from a learning mechanism that rewards actions that have the ability to bring the system to states that complies with the defined goals.

As a basis for the optimal adaptation, required performance as well as prices and penalty agreements defined in the SLAs must be taken into consideration. Service income includes the estimated income paid by the users for using services in normal QoS conditions and the penalty cost paid back to the users when the service qualities and functionalities are lower than defined by SLA. In general, goal, policy and inherent state concepts have the SLA class as a parameter. The inherent states of the service components can comprise measures related to functionality, performance and income. The goal is defined by a goal expression and a weight. The goal expression defines a required system performance or service income measure. A goal example is: "Service response time of premium service SLA class < 2 secs". The goal weight identifies a goal's importance. A goal can be associated with a set of policies. A policy is defined by *conditions*, *constraints* and *actions*. The *condition* defines the activation of the policy execution. The *constraint* restricts the usage of the policy, and is described by an expression of required and inherent functionality and performance of services and capabilities, required and inherent service incomes, available nodes and their capabilities, as well as system time. A policy example related to the goal example given above is: "*If CPU utilization > 95% and the time is between 18:00-24:00, ignore new service requests of users of ordinary SLA classes that request service time > 2 mins*". It is expressed with Conditions: CPU utilization > 95%, Constraints: system time between 18:00-24:00 and service time request > 2 minutes, and Actions: ignore new service requests of users of ordinary SLA classes.

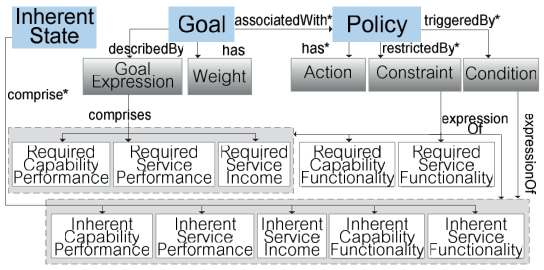

Inherent State
Goal
associatedWith*
Policy
triggeredBy*
describedBy
has
has*
restrictedBy*
Goal Expression
Weight
Action
Constraint
Condition
comprise*
comprises
expression Of
expressionOf
Required Capability Performance
Required Service Performance
Required Service Income
Required Capability Functionality
Required Service Functionality
Inherent Capability Performance
Inherent Service Performance
Inherent Service Income
Inherent Capability Functionality
Inherent Service Functionality

Figure 8. A Static Goal-based Policy Ontology

Table 1 lists notations used for capability, service and income concepts. The *required capability* functionality and performance are service component defined requirements. *Required service* functionality and performance are SLA defined requirements.

Table 1. Capability, service and income concepts notation

Notation	Description
$\hat{C}_R$	Required capability performance set
$\hat{C}_I$	Inherent capability performance set
$\overline{C_R}$	Required capability functionality set
$\overline{C_I}$	Inherent capability functionality set
$\hat{C}_{A,n}$	Set of available capabilities in node n; n=[1, N]
$\hat{S}_R$	Required service performance set
$\hat{S}_I$	Inherent service performance set
$\overline{S_R}$	Required service functionality set
$\overline{S_I}$	Inherent service functionality set
I_R	Required service income
I_I	Inherent service income

An RM model R extended from [3, 4] is now defined as:

$$R \equiv \{ \Theta, \Phi, \Pi, \xi \} \tag{1}$$

Here Θ is a set of query expressions with variables, Φ is a generic *reasoning procedure,* Π is a set of policies and ξ is the data including the inherent states values. The expression (1) can be further elaborated as follows:

$$\xi \equiv (\overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I, \hat{C}_{A,n}; n=[1, N]) \tag{2}$$

$$\Pi \equiv \{ p_i \} \tag{3}$$

$$p_i \equiv (\Sigma_i, X_i, A_i) \tag{4}$$

$$\Sigma_i \equiv \text{Expression}(\overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I) \tag{5}$$

$$X_i \equiv \text{Expression}(\overline{S_R}, \hat{S}_R, \overline{C_R}, \hat{C}_R, I_R, \overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I, \hat{C}_{A,n}; n=[1, N], \Gamma) \tag{6}$$

A policy p_i has conditions Σ_i, constraints X_i and actions A_i. The condition is an expression of the inherent states of relevant service components. The constraint is an expression of required functionality and performance of services and capabilities, required service incomes, relevant service components, available nodes and their capabilities, as well as the system clock time Γ.

A reasoning procedure is applied to select appropriate actions with maximum accumulated rewards. It is based on *Equivalent transformation (ET)* [21], which solves a given problem by finding values for the variables of the queries. The conditions, constraints and actions can have variables. The result of the reasoning procedure can, in addition to actions, give instantiated variables.

7.2. A Dynamic Model based on a Learning Mechanism

The static model presented in the previous subsection can be made dynamic by introducing a learning mechanism (LM) and by adding the parameters *Accumulated Reward* and *Operation Cost* in the model as illustrated in Figure 9.

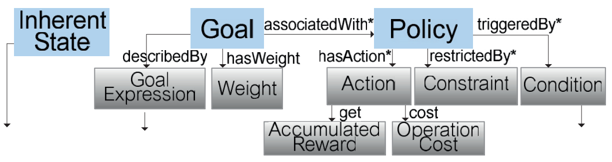


Figure 9. Making a Dynamic Goal-based Policy Ontology.

The proposed LM model gives rewards to actions to be selected by RM. The reward is measure for the ability *to move towards a state with goal performance and income measures.* The rewards will be accumulated over a period of time. The *LM* model L is defined as:

$$L \equiv \{ \Omega, \Lambda, \Psi, \zeta \} \tag{7}$$

where Ω is a set of *goals*, Λ is a generic *rewarding procedure*, Ψ is a *reward database* storing the accumulated rewards of actions, and ζ is the LM data including the inherent states from this service component as well as other service components. We further have:

$$\zeta \equiv (\overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I) \tag{8}$$

$$\Omega \equiv \{ g_k \} \tag{9}$$

$$g_k \equiv (d_k, \omega_k) \tag{10}$$

A goal g_k has goal expression d_k and weight ω_k. The sum of the goal weights is equal to 1. At time t, the rewarding procedure will calculate the reward of an action a_i, which was applied at time t-1 as:

$$\text{reward}(a_i,i_{k,t-1},d_k) = (\Delta(i_{k,t},i_{k,t-1})/\Delta(d_k,i_{k,t-1})) * \omega_k - \text{cost}(a_i) \tag{11}$$

where $i_{k,t-1}$ and $i_{k,t}$ are an inherent state measure before and after applying the action for an monitoring interval [t-1, t], $i_k \in \zeta$ and d_k is an associated goal required measures. $\Delta(i_{k,t},i_{k,t-1})$ is the difference between $i_{k,t}$ and $i_{k,t-1}$. $\Delta(d_k,i_{k,t-1})$ is the difference between d_k and $i_{k,t-1}$. ω_k is the goal weight and cost(a_i) is the operation cost of a_i. The measure accumulated_reward(a_i,$i_{k,t-1}$,d_k), is then the sum of the rewards of an action a_i for an inherent state measure $i_{k,t-1}$ and a goal measure d_k. Equation (2) is accordingly modified as follows to include the reward database Ψ:

$$\xi \equiv (\overline{S_I} , \hat{S}_I, \overline{C_I} , \hat{C}_I, I_I, \Psi, \hat{C}_{A,n}; n=[1, N]) \qquad (2')$$

8. Case Study

An example music video streaming system is presented with the intention to demonstrate the *Reasoning machine* and *Learning mechanism* behavior models as presented in Section 7. The streaming system is illustrated in Figure 10. The streaming case is the same as applied in a previous work [4]. The goal-based policy and the learning mechanism, however, were not applied in [4].

The system is constituted by the following service components implementing the service functionalities as defined in Section 6: *Capability and Service Administration (CSA)*, *Capability and Service Monitoring (CSM)*, *Fault Diagnosis (FD)*, *Capability Configuration (CC)*, *Service Installation and Instantiation* (*SII*), *Service Component Movement (SM*) and *Primary Service Components (PSC)*.

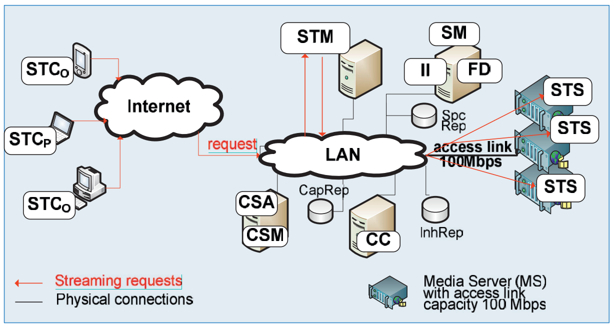


Figure 10. An Example Streaming System

In accordance with the previous discussion of boundary between primary service functionality and service management functionality, *Capability Usage Allocation* as well as *Performance Diagnosis* as defined in Section 4, is now realized by PSCs. The PSCs are S*treaming Client (STC)*, *Streaming Manager (STM)* and *Streaming Server (STS)*. An STS, executing on a *media streaming server (MS)*, streams the music video files to STCs. STM will accept the streaming requests on behalf of STSs. STM will decide which STS that can serve the requests, or STM may put them in waiting queues. STM can also instantiate a new STS in an available MS without executing STS.

An STC is associated with an SLA class, which defines *required streaming throughput*, *price for the service* and *service provider penalties* if the agreed QoS cannot be met. Two SLA classes are applied: *premium (P)* and *ordinary (O)*. An STC is denoted by its SLA class as *STC_P* or *STC_O*. Each SLA class has different required throughput (X); the STC_P required throughput (X_P) can be 1Mbps or 600Kbps for high-resolution and degraded fair-resolution videos, while the STC_O required throughput (X_O) is 500Kbps for low-resolution videos. The MS's required access link capacity ($C_{R,AL}$) is set to 100 Mbps. The number of STCs that can use the service at a time is limited by the MS access link capacity. When the required streaming throughput cannot be provided, a STC needs to wait until some streaming connections have finished. An STC_O can be disconnected, while an STC_P may have to degrade the video resolution.The service provider will pay penalties in case of waiting and disconnection of the STC.

The penalty and price functions are given in Table 2. A cost unit is the price paid by an ordinary client for one second streaming of the rate 500Kpbs. The price function for using the service is M(SLA_Class,X) (cost units/second). The penalty function for waiting is P_{WAIT}(SLA_Class) (cost units/second), and the penalty function for disconnection is P_{DISC}(SLA_Class) (cost units/connection).

Table 2. Prices and penalty functions

	STC_O (X_O=500Kbps)	STC_P (X_P=600Kbps)	STC_P (X_P=1Mbps)
M(SLA_Class,X)/s	1	1.875	2
P_{WAIT}(SLA_Class)/s	5	10	10
P_{DISC}(SLA_Class)/Connection	10	-	-

The complete set of actions A = {a_D, a_B, a_N, a_I, a_R, a_T, a_M} and a subset Á=A–{a_M}.The action a_D disconnects the ordinary clients, a_B decreases the throughput of the premium clients. The action a_N instantiates a MS, a_I instantiates a new STS, a_R disconnects a MS, a_T terminates an STS and a_M moves connected client sessions from an STS to another STS. These actions are selected by the Reasoning machine of STM. STM executes a_N, a_I, a_R and a_T, while STM suggests a_D, a_B and a_M to STSs.

The considered capability is the MS access link. The required and inherent capability performance sets are denoted as $\hat{C}_R \equiv \{C_{R,AL}\}$ and $\hat{C}_I \equiv \{C_{I,AL}\}$, where $C_{R,AL}$ is the required access link capacity, and $C_{I,AL}$ is the available access link capacity. The inherent service performance set $\hat{S}_I$ 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 ordinary clients ($N_{Disc,O}$), the number of MS (N_{Node}), the service time and waiting time of premium and ordinary clients ($T_{Serv,P}$, $T_{Serv,O}$, $T_{Wait,P}$, $T_{Wait,O}$). These values as well as the inherent service income (I_I) are observed per a monitoring interval Δ. The service income is defined as:

$$I_I = M(STC_O,X_O)*T_{Serv,O} + M(STC_P,X_P)*T_{Serv,P} - P_{WAIT}(STC_O)*T_{Wait,O} - P_{WAIT}(STC_P)*T_{Wait,P} - P_{DISC}(STC_O)*N_{Disc,O} - P_{Ser}*N_{Node}*\Delta \quad (13)$$

where P_{Ser} is the cost function for adding a new MS which is 150 units/second per node, while M(SLA_Class,X), P_{WAIT}(SLA_Class) and P_{DISC}(SLA_Class) are as already defined in Table 2.

8.1. RM and LM Specification

In this case study, STM plays an important role. Its RM specification is defined as follows:

$$R_{STM} \equiv \{ \Theta_{STM}, \Phi, \Pi_{STM}, \xi_{STM} \} \quad (14)$$

Π_{STM} consists of five policies (p_1-p_5) as presented in Appendix. The LM applied by STM is defined as follows:

$$L_{STM} \equiv \{ \Omega_{STM}, \Lambda, \Psi_{STM}, \zeta_{STM} \} \tag{15}$$
$$\Omega_{STM} \equiv \{ g_1, g_2\} \tag{16}$$
$$g_1 \equiv (d_1: I_R > 0, \omega_1: 0.8) \tag{17}$$
$$g_2 \equiv (d_2: T_{Wait} < \Delta, \omega_2: 0.2) \tag{18}$$

Here I_R is the required service income, and T_{Wait} is the sum of the waiting time of premium and ordinary clients. These goals are set in order to gain high income and to avoid high waiting time. The policies p_1-p_5 can be used when the required service income is not met, while the policies p_1-p_3 are used when the waiting time is higher than expected.

8.2. Experiments and Results

The measures considered are *accumulated service income* and the *accumulated waiting time*. The streaming request arrivals are modelled as a Poisson process with an arrival intensity parameter λ_{SLA_Class}. The duration of streaming connections d_{SLA_Class} is constant and is set to 10 minutes. The traffic per MS access link ρ is defined as:

$$\rho = ((\lambda_P{*}d_P{*}X_P) + (\lambda_O{*}d_O{*}X_O))/ (N_{Node}{*}C_{I,AL}) \tag{19}$$

The monitoring interval Δ is 1 minute. The STCs will stop waiting, and there is no penalty for waiting after 10 minutes. The number of available MS = 3. Initially, only one STS in one MS is instantiated. Three cases are considered:

I. The complete set of actions A is used,
II. The action subset Á is used.
III. Action set is A as in Case I, but no *Learning mechanism is used.* So the actions are not rewarded.

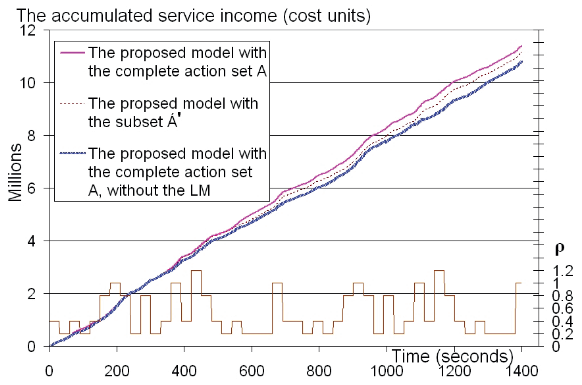


Figure 11. Accumulated service income.

Figure 11 and 12 show the accumulated service income and the accumulated waiting time of three cases I, II and III. The traffics offered are a function of time. The time with ρ at a fixed level, denoted as the *ρ period*, is set to 30 minutes. ρ varies from 0.2 to 1.2. λ_P is set to 50% of the total arrival intensity.

The brown line in these figures shows the variation of ρ. In Case I, the system learned that {a_M, a_T and a_R}, which move connected STC sessions, terminate an STS and disconnect a MS consecutively, is efficient to adapt the system when ρ drops and then the required service income is not met. As a result, Case I could produce the highest accumulated service income and the lowest accumulated waiting time. For the last case, the actions were selected randomly and they were not appropriate to the states of unwanted service income and the waiting time. So, the accumulated service income of Case III was the lowest, while the accumulated waiting time was the highest. So the learning mechanism applied has positive influence on both service income and waiting time.

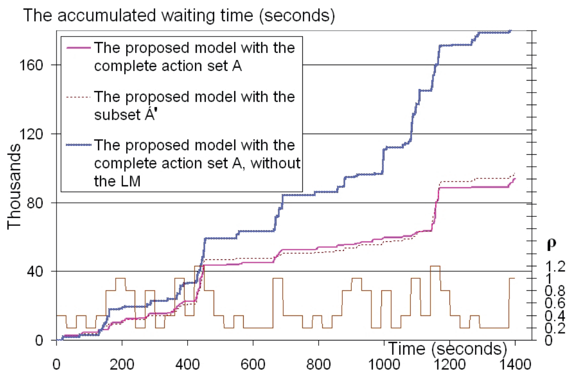


Figure 12. Accumulated waiting time.

9. Summary and Conclusions

Issues of adaptability of networked service systems both within a general and a scoped view have been presented. The general view considers issues of adaptation at two levels: 1) System of entities and functionalities related to the service system life-cycle, and adaptability types with required adaptability functionality, and 2) Architectures supporting adaptability. The adaptability types defined are capability-related, functionality-related and context-related adaptation. The architecture supporting adaptability is constituted by a computing architecture and service functionality architecture. The computing architecture has two layers represented by a service layer and a physical layer. The adaptability functionality is realized by Actors interpreting EFSM specifications supported by a Reasoning machine and a Learning mechanism. The presented computing architecture has functionality and performance concepts both related to capability and service. This provides a basis for basic capability-related awareness, which is the basis for all adaptation types as defined in Section 5. The architecture further opens for service systems defined by flexible combination of EFSM-, goal-, and policy specifications combined with learning mechanisms. The two-level architecture combined with the flexible Actor execution behavior also makes adaptation during runtime possible. This basic model should make it possible to define and implement all adaptation functionalities as defined in Section 5. Concerning context adaptation, the model is open to the use of sensor capabilities. In specific application cases, refinement and elaboration is needed.

The scoped view considers capability-related adaptation. A goal-based policy ontology was presented. It is realized by EFSMs, reasoning machine and learning mechanism. Finally a case study was finally presented to demonstrate the use and efficiency of the goal-based policy ontology as well as its realization by concrete policies and learning algorithm. Performance measures considered are service income and waiting time. For the considered cases, the learning mechanism has positive influence on the performance measures considered.

REFERENCES

[1] J. O. Kephart and D. M. Chess. “The Vision of Autonomic Computing”, IEEE Computer Society, January 2003, pp. 41-47.

[2] S. White, J. Hanson, I. Whalley, D. Chess, and J. Kephart. “An architectural approach to autonomic computing”, In Proc. of 1st IEEE Int. Conf. on autonomic computing, New York, May 2004, pp. 2–9.

[3] P. Supadulchai and F. A. Aagesen. “Policy-based Adaptable Service Systems Architecture”, In Proc. of 21st IEEE Int. Conf. on Advanced Information Networking and Applications (AINA’07), Canada, 2007.

[4] P. Supadulchai, F. A. Aagesen and P. Thongtra. “Towards Policy-Supported Adaptable Service Systems”, EUNICE 13th EUNICE and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services. Lecture Notes in Computer Science (LCNS) 4606, pp 128-140.

[5] P. Thongtra and F. A. Aagesen. “An Adaptable Capability Monitoring System”, In Proc. of 6th Int. Conference on Networking and Services (ICNS 2010), Mexico, March, 2010.

[6] J. C. Strassner, N. Agoulmine, and E. Lehtihet. “FOCALE - A Novel Autonomic Networking Architecture”, Latin American Autonomic Computing Symposium (LAACS), 2006, Campo Grande, MS, Brazil.

[7] K. McCloghrie, et al. RFC 2578, Structure of Management Information Version 2 (SMIv2), April 1999.

[8] A. Westerinen and J. C. Strassner. “Common Information Model (CIM) Core Model”, Version 2.4, Distributed Management Task Force White Paper DSP0111, August 2000.

[9] P. Thongtra and F. A. Aagesen. “Capability Ontology in Adaptable Service System Framework”, In Proc. of 5th Int. Multi-Conference on Computing in the Global Information Technology, Spain, Sep 2010.

[10] P. Liljeback. *Modelling, Development and Control of Snake Robots*, PhD Thesis at NTNU, 2011, ISBN 978-82-471-2667-7.

[11] H. Skubch, M. Wagner, R. Reichle, and K. Geihs. “A modelling language for cooperative plans in highly dynamic domains”, Elsvier Mechatronics 21 (2011) 423–433.

[12] K. Geihs, C. Evers, R. Reichle, M. Wagner, and M. U. Khan. “Development Support for QoS-Aware Service-Adaptation in Ubiquitous Computing Applications”, SAC’11, March 21-25, 2011, TaiChung, Taiwan, ACM 978-1-4503-0113-8/11/03.

[13] R. Di Cosmo, D. Di Ruscioa, P. Pelliccionea, A. Pierantonioa, and S. Zacchiroli. “Supporting Software Evolution”, Component-Based FOSS Systems, Science of Computer Programming Volume 76, Issue 12, 1 December 2011, pp. 1144-1160, Special Issue on Software Evolution, Adaptability and Variability.

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

[15] S. van der Meer, et al. “Autonomic Networking: Prototype Implementation of the Policy Continuum”, In Proc. of 1st IEEE International Workshop on Broadband Convergence Networks (BCN 2006), pages 1-10, IEEE, New York, 2006.

[16] R. Enns. RFC 4741, Network Configuration Protocol (NETCONF).

[17] M. Bjorklund. RFC 6020, YANG - A Data Modelling Language for the Network Configuration Protocol (NETCONF), October 2010

[18] W3C, OWL Web Ontology Language Overview, 2004. Available at: http://www.w3.org/TR/owl-features/

[19] V. Wuwonse and M. Yoshikawa, “Towards a language for metadata schemas for interoperability”, In Proc. of 4th Int. Conf. on Dublin Core and Metadata Applications, China, 2004.

[20] R. Studer, V. R. Benjamins, and D. Fensel, “Knowledge Engineering: Principles and methods,” Data & Knowledge Engineering, vol. 25, pp. 161-197, 1998.

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

[22] F. Berman, et al. “Adaptive computing on the grid using AppLeS”, IEEE Trans. Parallel Distributed System, vol. 14, no. 4, pp. 369–382, Apr. 2003.

[23] P. Boinot, R. Marlet, J. Noy´e, G. Muller, and C. Cosell. “A declarative approach for designing and developing adaptive components”, In Proc. of the 15th IEEE Int. Conf. on Automated Software Engineering, 2000.

[24] H. Liu and M. Parashar. “Accord: a programming framework for autonomic applications”, In IEEE Trans. on System, Man, and Cybernetics, vol. 36, pp. 341–352, 2006.

[25] L. Lymberopoulos, E. C. Lupu and M. S. Sloman. “An Adaptive Policy-Based Framework for Network Services Management”, In Journal of Networks and Systems Management, vol. 11, pp. 277–303, 2003.

[26] K. Yoshihara, M. Isomura, and H. Horiuchi. “Distributed Policy-based Management Enabling Policy Adaptation on Monitoring using Active Network Technology”, In Proc. of 12th IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management, France, Oct 2001.

[27] G. Tesauro, R. Das, N.K. Jong, and M.N. Bennani. “A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation”, In Proc. of 3rd IEEE Int. Conf. on Autonomic Computing (ICAC’06), Ireland, Jun 2006, pp. 65–73.

[28] M. Mesnier, et al. “File classification in self-* storage systems”, In Proc. of Int. Conf. on Autonomic Computing (ICAC-04), pp. 44–51.

APPENDIX: POLICY SPECIFICATIONS

The five policies (p_1-p_5) used in the case study are specified below by OWL [18] and OWL/XDD [19]. Variables are integrated with ordinary OWL elements and are prefixed with $.

p_1	**Conditions:** $\$I_I <= 0$ or $\$T_{Wait} >= \Delta$, **Constraints:** $P_{WAIT}(STC_O) < P_{WAIT}(STC_P)$, **Actions:** $\{a_D\}$ **Operation Cost:** a_D costs $P_{DISC}(STC_O)$ units. This policy can be read as: a_D should be used to disconnect a list of STC_O when $P_{WAIT}(STC_O) < P_{WAIT}(STC_P)$, and the number of STC_O being disconnected is calculated from $X_{P,1Mbps} * \$N_{Wait,P} / X_O$.
p_2	**Conditions:** $\$I_I <= 0$ or $\$T_{Wait} >= \Delta$, **Constraints:** $P_{WAIT}(STC_O) > M(STC_P,X_{P,1Mbps})-M(STC_P,X_{P,600Kbps})$, **Actions:** $\{a_B\}$, **Operation Cost:** a_B costs $M(STC_P,X_{P,1Mbps})$ - $M(STC_P,X_{P,600Kbps})$ This policy can be read as: a_B should be used to decrease the throughput of a list of STC_P when $P_{WAIT}(STC_O) >$

	$M(STC_P, X_{P,1Mbps})$ - $M(STC_P, X_{P,600Kbps})$, and the number of STC_P to decrease the throughput is calculated from $X_O * \$N_{Wait,O} / (X_{P,1Mbps} - X_{P,600Kbps})$.
p_3	**Conditions:** $\$I_I <= 0$ or $\$T_{Wait} >= \Delta$, **Constraints:** $(X_{P,\ 1Mbps} * \$N_{Wait,P} + X_O * \$N_{Wait,O}) / C_{R,AL} > 0.1$, **Actions:** $\{a_N, a_I\}$, **Operation Cost:** The actions $\{a_N, a_I\}$ cost $P_{Ser} * \Delta$ This policy can be read as: a_N and a_I should be used to instantiate a MS and to instantiate a new STS consecutively, when $(X_{P,\ 1Mbps} * \$N_{Wait,P} + X_O * \$N_{Wait,O}) / C_{R,AL} > 0.1$.
p_4	**Conditions:** $\$I_I <= 0$, **Constraints:** $(X_{P,\ 1Mbps} * \$N_{Wait,P} + X_O * \$N_{Wait,O}) / C_{R,AL} < 0.1$, **Actions:** $\{a_T, a_R\}$, **Operation Cost:** The actions $\{a_T, a_R\}$ cost $P_{DISC}(STC_O) + P_{WAIT}(STC_P) – P_{Ser} * \Delta$ This policy can be read as: a_T and a_R should be used to terminate an STS and to disconnect a MS consecutively, when $(X_{P,\ 1Mbps} * \$N_{Wait,P} + X_O * \$N_{Wait,O}) / C_{R,AL} < 0.1$.
p_5	**Conditions:** $\$I_I <= 0$, **Constraints:** $(X_{P,\ 1Mbps} * \$N_{Wait,P} + X_O * \$N_{Wait,O}) / C_{R,AL} < 0.1$, **Actions:** $\{a_M, a_T, a_R\}$, **Operation Cost:** These actions $\{a_M, a_T, a_R\}$ make profit $P_{Ser} * \Delta$ This policy can be read as: a_M, a_T and a_R should be used to move connected STC sessions, to terminate an STS and to disconnect a MS consecutively, when $(X_{P,\ 1Mbps} * \$N_{Wait,P} + X_O * \$N_{Wait,O}) / C_{R,AL} < 0.1$.

Authors Short Biography

Finn Arve Aagesen is Professor at the Department of telematics at NTNU (Norwegian University of Science and Technology), Norway. He has a PhD in Communication Technology from NTNU in 1977. He is the Norwegian delegate to IFIP TC6 Communication since 1996. His general research interests include network and service management, systems engineering and adaptable systems. His recent research interest also includes platforms for Smart Grid control systems.

Patcharee Thongtra received the B.E. degree in computer engineering from Chulalongkorn University, Thailand, in 1999, and the M.Sc. degree in computer science from Asian Institute of Technology, Thailand, in 2002. Currently, she is a PhD candidate at Department of Telematics, NTNU (Norwegian University of Science and Technology), Norway. Her research interests include adaptable systems, autonomic computing, network and service management, semantic web, and software engineering.