A Capability-based Service Framework for Adaptable Service Systems

Finn Arve Aagesen and Paramai Supadulchai
Institute of Telematics, NTNU
N7491 Trondheim, Norway
finnarve@item.ntnu.no, paramai@item.ntnu.no

Abstract

This paper presents a capability-based service framework for adaptable service systems. The paper has focus on: I) Adaptability properties, II) The architecture solution needed to meet these properties, and III) Capability configuration management functionality.

Adaptability is based on flexibility. Accordingly, basic rearrangement flexibility must be provided. But the framework must in addition have necessary concepts, features and functionality that make it possible to adapt to various traffic situations and failure states.

The architecture solution has a computing and a service functionality dimension. The computing architecture is based on a theatre metaphor, where actor, manuscript, role figure and capability are core concepts. Fundamental QoS concepts are capability, capability performance, service performance and service level agreements. Actors, which are the basis for the implementation of the service functionality, can be Extended Finite State Machines or Reasoning Machines. Actors are able to download and execute EFSM- and RM-based manuscripts.

Models for capability configuration management functionality are presented. Reasoning machines are used for capability configuration management in general as well as for policy adaptation.

1. Introduction

Networked service systems are considered. Services are realized by service components, which by their inter-working provide a service in the role of a service provider to a service user. Service components are executed as software components in nodes, which are physical processing units such as servers, routers, switches, PCs and mobile phones. A *service framework* is here defined as the overall structural and behavior framework for the specification and execution of services. Networked services have during more than two decade been is an important research topic. Topics include Intelligent Networks and TINA (Tele-communication Information Networking Architecture) [1], Mobile Agents and Active and Programmable Networks [2] in the 80-ies and 90-ies. Focus was on flexibility and efficiency in the definition, deployment and execution. This focus has now been changed into *adaptability* and *evolution*. We have entered an era with a high degree of flexibility. To utilize the flexibility potential as a foundation for adaptability, the attributes of services and nodes must be appropriately formalized, stored and made available.

An adaptable service system is here defined as a service system which is able to adapt dynamically to changes in time and position related to users, nodes, capabilities, system performance, changed service requirements and policies. In this context, capability is defined as an inherent physical property of a node, which is used as a basis for implementing services (See Section 2). This definition of adaptability covers a wide spectrum of functionality, and does also include autonomic communication [3, 4].

This paper presents some issues related to service frameworks for adaptability. The issues presented are part of *TAPAS* (TAPAS = Telematics Architecture for Play-based Adaptable Systems. See [5] and the URL: http://tapas.item.ntnu.no. The paper has focus on three issues: I) Adaptability properties, II) Architecture solution and III) Capability configuration management functionality

The adaptability properties presented in Section 2 are the basis for the architecture solution which is presented in Section 3-4. The architecture has a computing dimension and a service functionality dimension. The computing architecture, which is based on a theatre metaphor, is presented in Section 5. An actor playing a role is either an Extended Finite State Machines (EFSM) or a Reasoning Machines (RM).

Issue III is presented in Section 5-6. Section 5 presents generic EFSM and RM models. Section 6 defines RM models for capability configuration management functionality. Related works are

presented in Section 7, while Section 8 gives Summary and Conclusions.

1. Adaptability Properties

General and core functional adaptability properties are defined. Four *general properties are defined.* The framework must provide a flexible and common way of modeling services independent of the type of the service system, the service models must be based on mechanisms appropriate for the type of functionality, the software implementation mechanisms must be flexible and powerful, and there must be easy mapping of the conceptual service system models to the physical computing and communication platform. The *core functional adaptability properties* are grouped in three classes: A): Rearrangement flexibility B): Failure robustness, and C): Resource load awareness and control

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

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

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

Property A defines the basic flexibility features of the architecture. Property B and C set requirements to concepts and features that are needed as a foundation for the specification and implementation of functionality that can provide these properties.

2. Architecture Solution – Features and Concepts

2.1 Service and computing architecture dimensions

To meet the *general properties defined in Section 2,* the TINA architecture principle [2] is followed. The TAPAS service framework has a computing architecture dimension and a functionality architecture dimension. *The service architecture* shows the structure of services and service components. The *computing architecture* is a generic architecture for the specification and execution of any service. While the service architecture has focus on the service functionality independent of implementation, the computing architecture has focus on the modeling of functionality with respect to implementation, but independent of the nature of the service functionality. The properties of the computing architecture, however, are the fundament for the creation of services with needed adaptable networking services properties. *So the core functional properties are realized by a mixture of the features and concepts of the computing architecture and the functionality of the service architecture.*

2.2 QoS related concepts

2.2.1 Capability and capability performance

Capability was introduced in Section 1. Capability is needed to meet all the three core properties. Performance concepts, however, are needed to meet the failure robustness as well as resource load awareness and control core property. *Capabilities* can be classified into resources, functions and data. Examples are CPU, memory, transmission links, special hardware, and special programs and data.

Capability performance measures comprise: i) capability capacity measures, ii) capability state measures and iii) capability QoS measures, i.e. traffic and dependability measures. *Capability capacity measure* examples are transmission channel capacity, CPU processing speed and disc size. *Capability state measure* examples are number of connections, and the number that is waiting. Important *traffic performance measures* are transfer time, waiting time, throughput and utilization. Important *dependability performance measures* are availability and recovery time.

2.2.2 Service performance

Service performance measures are performance concepts related to the service provided to the service user. These measures comprise i) service system state measures and ii) service system QoS measures, i.e. traffic and dependability measures

2.2.3 Service level agreements

Service level agreements (SLA) are agreements between the *service users* and the *service provider*.

The agreement can contain elements such as: required capabilities, required service performance, payment for the service when the agreed QoS is offered and payment for the service in case of reduced QoS.

3. Computing Architecture

3.1 The theatre metaphor

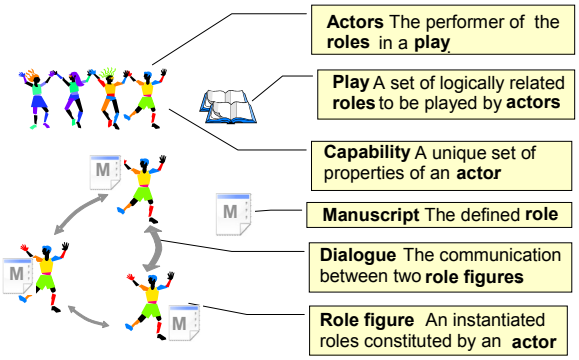

Figure 1 - The theatre metaphor

The computing architecture is founded on a theatre metaphor (Figure 1). Actors perform roles according to manuscripts. Actors are software components in the nodes that can download manuscripts defining the roles to be played. An actor will constitute a *role figure* by behaving according to a manuscript. A *play* consists of several *actors* playing different *roles*, each possibly having different requirements on *capabilities*.

3.2 Computing architecture viewpoints

The computing architecture is illustrated in Figure 2. For simplicity, the service user and service level agreements (SLA) are not included in the figure. The architecture has three views: the *service* view, *play* view and *physical* view. The service view considers the service as constituted by conceptual service components. The leaf conceptual service components are constituted by role figures which are implemented by actors. The actors are executed as operating system software components in nodes, which are physical processing units such as servers, routers, switches and user terminals.

The three view model is not a layered service model. A service system and its service components have models and model parameters related to the service view, play view and physical view. Capability and capability performance measures defined in the physical view, however, are visible in the play view as well as the service view. The service performance measures and SLA defined in the service view are also visible in the play view. This is not explicitly illustrated in Figure 2.

The *core platform* is a platform supporting the execution of service functionality based on the computing architecture functionality.

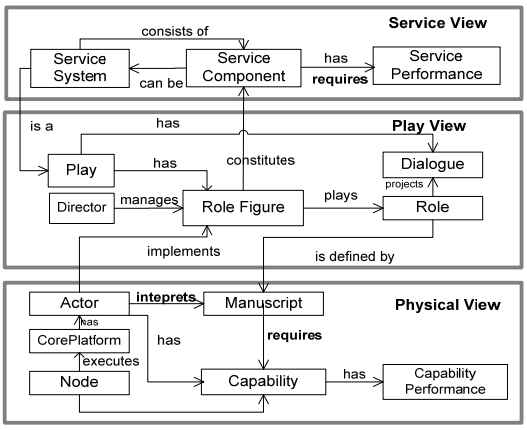

Figure 2 - TAPAS computing architecture

The computing architecture concepts and functionality are intended to be a basis for designing functionality that can meet the *general* as well as the *core properties*. The three views meet the general architecture requirements. Concerning the core properties, the play view concepts are primarily rearrangement flexibility oriented. The QoS concepts, gives a basis for functionality that can meet the *robustness and survivability* as well as the *QoS awareness and resource control* properties.

3.3 Actor and role figure types

An actor can be an EFSM or RM (See Section 1). Actors are able to download and execute EFSM- and RM-based manuscripts. Networked services are traditionally based on EFSM-based specification and execution. RM makes policy-based specification and operation possible. Policy-based software has a specification style, which is expressive and flexible.

A policy is a set of rules with related actions. A *policy system* is a set of policies. An RM is using a policy system to manage the behavior of a target system. A *static policy system* has a non-changeable set of policies, while a *dynamic policy* system has a changeable set of policies.

There are two types of role figures: i) EFSM-based role figures and ii) RM-based role figures. The EFSM-based role figure is constituted by an EFSM actor. The RM-based role figure is constituted by an RM actor.

4. Capability Configuration Management

Executing service components are instances of service component types. Service components constituted by EFSM-based and RM-based role figurers are denoted as *EFSM-based* and *RM-based* service components, respectively. EFSM-based service components will have requirements with respect to capability and service performance to perform their intended functionality (Figure 3).

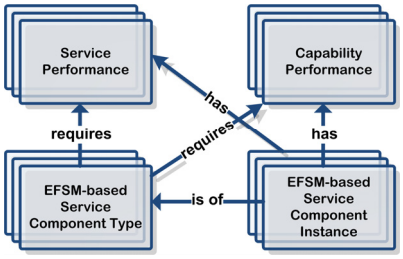

Figure 3 – EFSM-based part of service system

These requirements are denoted as *required* capability and service performance. The capabilities and service performance of an executing service system are denoted as *inherent* capabilities and service performance. *System performance* is now defined as the sum of capability performance and service performance.

Capability configuration management is defined as the allocation, re-allocation and de-allocation of capabilities and comprises: i) service system capability initialization and re-initialization and ii) capability allocation adaptation. *Service system capability initialization* and re-initialization is the allocation of the capabilities for the service components to be distributed and instantiated. *Capability allocation adaptation* is the monitoring of the performance of the executing service system and the reallocation of capabilities within the executing service systems. In situations when the instantiated service systems are unable to adapt satisfactorily, capability configuration management can initiate a service system *capability re-initialization* for a re-distribution and re-instantiation of the service system(s).

The service functionality architecture will need a rich set of functionality components to fulfill the core properties A)-C). On one hand, this paper has focus on the computing architecture features and concepts needed as basis for the general and core adaptability properties. On the other hand the paper has focus on the capability configuration management, which is an important adaptability function needed to utilize the adaptation potential of the capability-based computing architecture.

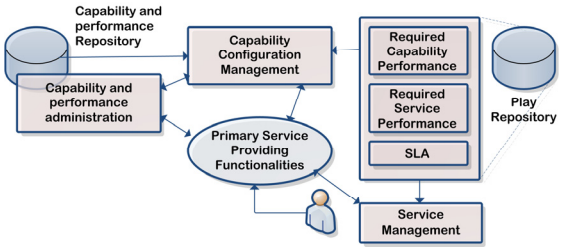

Figure 4 – Some components of the service functionality architecture

The main components of the service functionality architecture related to capability configuration management are illustrated in Figure 4. In addition to the *primary service provisioning functionality*, there must be functionality for management of the networked service systems. This *management functionality* comprises: i) *service management. i.e.:* service specification, implementation, deployment, invocation, exhibition, discovery and access, ii) *capability and performance administration, i.e.:* monitoring and registration of available and allocated capabilities, monitoring and registration of system performance, and the provisioning of a view of available capabilities, capability performance and system performance, and iii) *capability configuration management,* with functionality as defined above.

5. Service Component Models

5.1 General

This section presents service component models. Section 5.2 presents generic EFSM and RM models. Section 5.3 presents the reasoning procedure applied by reasoning-based service components. The application domain for reasoning based service components and the integration of EFSMs and RMs in reasoning clusters are presented in Section 5.4.

5.2 EFSM and RM models

The following concepts are defined:

- E Functionality set of an EFSM type
- $\hat{E}$ Functionality set of an EFSM instance
- $\mathcal{R}$ Functionality set of a RM type
- $\hat{\mathcal{R}}$ Functionality set of a RM instance
- M Manuscript type (EFSM or RM type)
- $\hat{M}$ Manuscript instance (EFSM or RM instance)
- C Capability performance measures set
- $\hat{C}_R$ Required capability performance set of an EFSM-based service component type
- $\hat{C}_I$ Inherent capability performance set of an executing EFSM-based service component type

$\hat{C}_A$ Set of available capabilities in nodes
S Service performance measures set
$\hat{S}_R$ Required service performance set for a EFSM-based service component type
$\hat{S}_I$ Inherent service performance set of an executing EFSM-based service component
I Income functions set for the service components constituting a service. These functions will depend on the system performance.

An EFSM type E is defined (≡) as:

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

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

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

where $\mathcal{Q}$ is the set of messages, $\mathcal{F}$ is a generic *reasoning procedure*, $\mathcal{P}$ is a policy system which consists of a set of *rules* $\mathcal{X}$ and a set of *actions* $\mathcal{A}$, $\mathcal{T}$ is a set of *system constraints* and $\mathcal{E}$ is a set of *facts*. The *reasoning procedure* is the procedure applied by RM to select the appropriate actions. Σ is a set of reasoning conditions defined by *trigger conditions* Σ_T, *and goal conditions* Σ_G. RM functionality is activated when a Σ_T is detected. When a trigger condition is true, the reasoning procedure transforms $\mathcal{Q}_1$ to $\mathcal{Q}_n$ by using $\mathcal{P}$ to match the system constraints $\mathcal{T}$ against the *facts* $\mathcal{E}$ and a set of suggest actions $\{\mathcal{A}_i, \mathcal{A}_j, \mathcal{A}_k \ldots\} \subseteq \mathcal{A}$.. The constraints rules and actions can have *variables*. The result of the reasoning can in addition to actions give instantiated variables.

The elements: $\mathcal{T}$, $\mathcal{E}$, Σ and $\mathcal{P}$ of an RM, depend on *the nature* of the reasoning service provided as well as the structural organization of EFSM, RM and capabilities. Section 5.4 will present reasoning models for cases that represents the functionality domain of the reasoning-based service components. These cases will be elaborated in Section 6.

5.3 Reasoning procedure

The reasoning procedure is based on *Equivalent Transformation* (*ET*) [6], which solves a given problem by transforming it through repetitive application of (semantically) equivalent transformation rules. Let $\mathbb{P}$ be a program which models an application for a knowledge base and $\mathcal{Q}_1$ is an initial message containing problems.

The semantics of $\mathbb{P} \cup \mathcal{Q}_1$ is denoted as $\mathcal{M}(\mathbb{P} \cup \mathcal{Q}_1)$. ET paradigm applies ET rules (procedural rewriting rules) in order to successively transform $\mathbb{P} \cup \mathcal{Q}_1$ into $\mathbb{P} \cup \mathcal{Q}_2$, $\mathbb{P} \cup \mathcal{Q}_3$, etc., while maintaining the condition $\mathcal{M}(\mathbb{P} \cup \mathcal{Q}_1) = \mathcal{M}(\mathbb{P} \cup \mathcal{Q}_2) = \mathcal{M}(\mathbb{P} \cup \mathcal{Q}_3) = \ldots$. Precisely, $\mathbb{P} \cup \mathcal{Q}_1$ is successively transformed until it becomes $\mathbb{P} \cup \mathcal{Q}_n$, where the message $\mathcal{Q}_n$ contains the list of *suggested actions* for the problem described by $\mathcal{Q}_1$. ET consists of sets of *ET rules* and ET *clauses* defined as follows:

ET clause: Head $\longleftarrow$ Body
ET rule: Head, Conditions $\longrightarrow$ Body

Head consists of one *head atom*, Body consists of body atoms and *Condition*s consists of condition atoms. A problem must be formulated as an ET clause. A *rule of a policy* as defined in Section 4.3 is modeled by an *ET rule*. The head atom is initially $\mathcal{Q}_1$. The *Head* will eventually contain $\mathcal{Q}_n$ if all the *body atoms* in the *Body* can be derived by *rules*. A body atom of a clause matching the head atom of a rule can be transformed into the rule's body atoms.

The reasoning procedure begins with a clause formulated by a message as follows:

$$\mathcal{Q}_1 \longleftarrow \mathcal{Q}_1 \quad (3)$$

The meaning of (3) is that the Head $\mathcal{Q}_1$ is true when the Body $\mathcal{Q}_1$ is true. The goal of the reasoning procedure is to transform (3) until no body atom is left. Consider the following rule (4):

$$\text{Head , Conditions} \xrightarrow{ET} B_1, B_2, \ldots B_n. \quad (4)$$

The rule (4) can transform the body atom $\mathcal{Q}_1$ of (3) into B_1, B_2, … B_n, provided that the body atom $\mathcal{Q}_1$ in (3) can match the Head in (4) and Conditions are not violated. Then, clause (3) will be transformed by (4) to (5) as follow:

$$\mathcal{Q}_2 \longleftarrow B_1', B_2', \ldots B_n'. \quad (5)$$

During the transformation, variables in $\mathcal{Q}_1 \ldots \mathcal{Q}_n$ and the *list of suggested actions*, which are a subset of the actions $\mathcal{A}$ as defined in (2b), will be instantiated. The transformation of a clause ends when either 1) no body atom of the clause is left or 2) no rule can transform the remaining body atoms of the clause.

5.4 *Functionality domain of reasoning machines*

The functionality domain of RM-based service components can be separated into two cases. In Case I, the RMs of a service system has no access to system performance data of the service system. In this case, the RM provides an ordinary procedural service only. In Case II, RMs of the service system has access to system performance data of the service system and takes part in the configuration management functionality and in close cooperation with one or more EFSMs. For this case RM can be used as:

A. a procedural reasoning service for an EFSM-based role figure involving the access to system performance data for service system capability initialization and re-initialization
B. a feed-back loop service involving one or more EFSM-based role figures with access to system performance data for capability allocation adaptation
C. a feed-back loop service involving one or more EFSM-based and RM-based role figures with access to system performance data for dynamic changes of the policies for capability allocation adaptation as defined in B) above.

A *reasoning cluster* is an independent unit with respect to reasoning. It is a collection of EFSM-based service components with an associated *reasoning system* constituted by one or more *RM-based service components*. A reasoning cluster has associated: $\mathcal{Q}$, $\mathcal{P}$, $\mathcal{T}$, $\mathcal{E}$ and Σ and consider them as common data among EFSM-based and RM-based role figures. An RM must be activated by an EFSM. Depending of the nature of the cluster and the nature of the reasoning, a dedicated EFSM denoted as E_Σ will be needed to inspect the reasoning conditions and activate or deactivate the RM-based role figures within a reasoning cluster. Three reasoning cluster types (A, B and C) corresponding to the cases A-C defined above are illustrated in Figure 5.

In the cluster type A, the RM is a procedure invoked by an EFSM-based role figure. The EFSM-based role figure can set the reasoning condition observed by E_Σ The cluster type B represents a close interaction between RM-based role figures ($\mathcal{R}$) and EFSM-based role figures (E). An E_Σ manages the activation and deactivation of EFSM-based role figures. In the cluster type C, an RM-based role figure manages another RM-based role figure's policies.

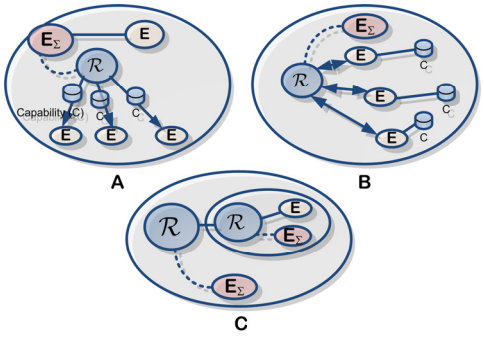

Figure 5 - Reasoning cluster types

6. Capability Configuration Management Functionality Models

In the following sections, reasoning models for the Cases A-C as defined in Section 6.4 will be elaborated from the generic RM type defined by (2a) and (2b) in Section 6.2.

6.1 *Capability (re-)initialization*

For this case, the indexes "init" and "re-init" indicates initialization and re-initialization. The constraints, facts and initialization condition can be expressed as follows:

$$\mathcal{T} \equiv Expr\{\, I, (\hat{S}_{R,k}, \hat{C}_{R,k};\ k = [1, K])\} \quad (6)$$

$$\mathcal{E}_{init} \equiv \{\, \hat{C}_{A,n};\ n = [1, N_{Node}]\,\} \quad (7)$$

$$\mathcal{E}_{re\text{-}init} \equiv \{((\hat{S}_{I,lk}, \hat{C}_{I,lk};\ l = [1, L_k]),\ k = [1, K]),\ (\hat{C}_{A,n};\ n = [1, N_{Node}])\} \quad (8)$$

$$\Sigma_{re\text{-}init} \equiv Expr\{\, \hat{S}_{R,k}, \hat{C}_{R,k};\ k = [1, K]\,\} \quad (9)$$

$\mathcal{T}$ is some expression containing the income functions, and the required capability and service performance measures $\hat{C}_{R,k}$ and $\hat{S}_{R,k}$. The element $\mathcal{E}_{init}$, which contains the facts for the capability initialization case, contains the set of unallocated capabilities $\hat{C}_{A,n}$ of N_{Node} that can potentially contribute their capabilities for the EFSM-based role figures. The element $\mathcal{E}_{re\text{-}init}$ contains the facts for the capability re-initialization case that consists of the set of inherent capabilities performance measures $\hat{C}_{I,lk}$, the set of inherent service performance measures $\hat{S}_{I,lk}$ as well as the set of available unallocated capability $\hat{C}_{A,n}$.

The reasoning condition for the capability initialization as well as the E_Σ is under the supervision of an EFSM-based role figure. The content of $\mathcal{P}$ depends on the needed behavior of the adaptable service system. The components constituting $\Sigma_{re\text{-}init}$ are expressions of the states and the variables of the required capability and service performance measures $\hat{S}_{R,k}$ and $\hat{C}_{R,k}$ as given in (8d). A total specification of initial configura-

tion and reconfiguration for a network printer example is given in [7].

6.2 Capability allocation adaptation

The constraints, facts and initialization condition can for this case be expressed similar to the equations for re-initialization (6), (8) and (9). The content of $\mathcal{P}$ depends on the needed behavior of the adaptable service system.

The feedback loop constituting a capability allocation adaptation is illustrated in Figure 6. An RM-based role figure called *Service System Adaptation Manager* $\mathcal{R}_1$ uses $\mathcal{E}$ as feedbacks from an EFSM-based role figure E. The rules $\mathcal{X}$ here are unchangeable. Uses of the expressions (6), (8) and (9) on a concrete capability allocation adaptation example using static policies are presented in [8].

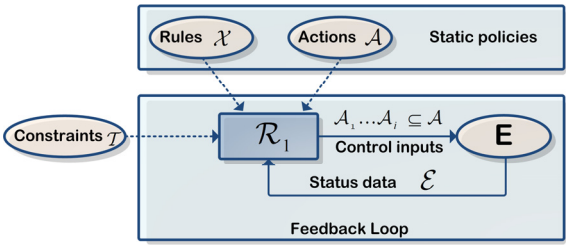

Figure 6 - Policy-based reasoning based on static policies

6.3 The adaptation of policies

The capability allocation adaptation using dynamic policies is illustrated in Figure 7. In addition to $\mathcal{R}_1$, a Policy Evaluator ($\mathcal{R}_2$) is added.

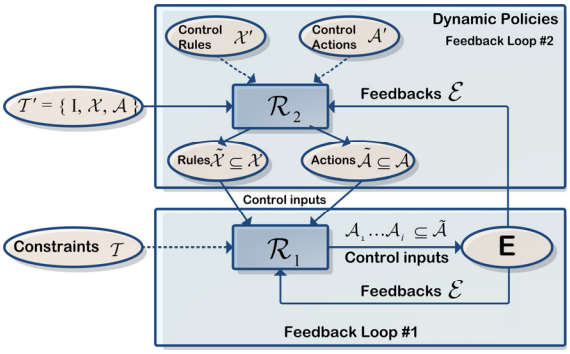

Figure 7 - Policy-based reasoning based on dynamic policies

A generic rule-based reasoning system with dynamic policies can be extended from (2a) and (2b) as follows:

$$\mathcal{R}_1 \equiv \{ \mathcal{Q}, \mathcal{F}, \tilde{\mathcal{P}}, \mathcal{T}, \mathcal{E}, \Sigma \} \quad (10)$$

$$\tilde{\mathcal{P}} \equiv \{ \tilde{\mathcal{X}}, \tilde{\mathcal{A}} \} \quad (11)$$

$$\mathcal{R}_2 \equiv \{ \mathcal{Q}', \mathcal{F}, \mathcal{P}', \mathcal{T}', \mathcal{E}', \Sigma' \} \quad (12)$$

$$\mathcal{P}' \equiv \{ \mathcal{X}', \mathcal{A}' \} \quad (13)$$

$$\mathcal{T}' \equiv \{ I, \mathcal{X}, \mathcal{A} \} \quad (14)$$

$\mathcal{Q}'$ is a set of internal messages between $\hat{\mathcal{R}}_1$ and $\hat{\mathcal{R}}_2$. The element $\mathcal{T}'$ consists of the set of income functions I and the set of total rules and actions ($\mathcal{X}$ and $\mathcal{A}$) of $\mathcal{R}_1$. The element $\mathcal{E}'$ is the same as $\mathcal{E}$ in 6.2. The element Σ' depends on the conditions for triggering the dynamic policies.

$\mathcal{P}'$ is a set of control policies consisting of the set of control rules $\mathcal{X}'$ and the set of control actions $\mathcal{A}'$. $\mathcal{X}'$ can use $\mathcal{A}'$ to re-order the priority, activate, de-activate and change the constraints of $\mathcal{X}$. As a result, the set of rules and actions used by $\mathcal{R}_1$ is now $\tilde{\mathcal{X}}$ and $\tilde{\mathcal{A}}$, which are derived from of $\mathcal{X}$ and $\mathcal{A}$.

The policy evaluator evaluates $\mathcal{X}$ at runtime based on *evaluation criteria, reference inputs* and *feedbacks*. Income functions are used as reference inputs, while the feedbacks are the system performance. Evaluation criteria can in general be history-based and prediction-based. The history-based evaluation determines the consequences of rules in the past based on system performance. The prediction-based evaluation determines the consequences of rules in the future based on mathematical equations represented by $\mathcal{X}'$. Dynamic policies need a certain period to evaluate the consequences of the rules used. Uses of the equations (10)-(14) on a concrete capability adaptation example using dynamic policies are presented in [8].

7. Related Work

Concerning service frameworks for adaptable service systems, focus is mostly on isolated features rather than on totality. Below we discuss related works that consider totality and that have rich feature sets.

The framework [9] has general architectural features and concepts and feedback loop for adaptability properties. However, it lacks a set of comprehensive features for capability configuration management. The work [10] has focus the capability allocation adaptation, but without considering general architectural features and concepts.

The architectures in [11] and [12] has a focus on agent-based computing architectures with feedback loops for autonomic service execution. However, it assumes that agents will have built-in service management functionality and all rely on some distributed algorithm. Likewise, [13] also share the same view while also describing a inclusive set of distributed protocols for adaptability properties.

Accord [14] is a framework with a comprehensive set of features including policies, and a computing architecture for service executions and configuration management. Physical capabilities, however, are not considered.

8. Summary and Conclusion

A capability-based service framework for adaptable service systems has been presented. The issues focused are I) Adaptability properties, II) Architecture solution and III) Capability configuration management functionality.

The *adaptability properties* defined are the basis for *architecture solution.* The *architecture* has a computing and a service architecture. The computing architecture has three views: the service view, the play view and the physical view. Further computing architecture features and concepts presented are:

- the theatre metaphor concepts (actor, role figure, capability, manuscript, role, play and dialogue)
- the ability of the Actor to download and interpret both EFSM-based and RM-based manuscripts
- QoS concepts (capability, service level agreement, capability and service performance)

Capability configuration management comprises service system capability initialization and re-initialization, and capability allocation adaptation. Generic models for EFSM-based and RM-based service components are presented. The application domain for reasoning machines within the context of capability configuration management are: a) capability initialization and re-initialization, b) capability allocation adaptation, and c) policy adaptation. RM-based service models for the cases a)-c) have been presented.

This paper, however, does neither contain data representation models nor concrete usage examples. The data representation of the QoS concepts, EFSMs and RMs are based on XML [8]. The aspects of capability configuration management that have been presented in this paper have been applied on concrete executable cases. A service system capability initialization and re-initialization example is presented in [7]. A capability allocation adaptation example is presented in [8].

References

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

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

[3] J. O. Kephart and D. M. Chess, "The Vision of Autunomic Computing," *Computer,* vol. 36, pp. 41-50, 2003.

[4] P. Horn, "Autonomic Computing: IBMs Perspective on the State of Information Technology", http://www.research.ibm.com/autonomic/.

[5] F. A. Aagesen, et al., "Towards a Plug and Play Architecture for Telecommunications," Proceedings of *IFIP TC6 Smartnet*, Bankok, 1999.

[6] K. Akama, et al., "Solving Problems by Equivalent Transformation of Declarative Programs," *Journal of the Japanese Society of Artificial Intelligence,* vol. 13, pp. 944-952, 1998.

[7] F. A. Aagesen, et al., "Configuration Management for an Adaptable Service System," Proceedings of *IFIP MAN*, Ho Chi Minh City, Viet Nam, 2005.

[8] P. Supadulchai and F. A. Aagesen, "Policy-based Adaptable Service Systems Architecture," Proceedings of *AINA-07*, Niagara Falls, Canada, 2007.

[9] D. Garlan, et al., "Rainbow: Architecture-Based Self-Adaptation with Reusable Instrastructure," *Computer,* vol. 37, pp. 46-54, 2004.

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

[11] D. Gracanin, et al., "Towards a model-driven architecture for autonomic systems," Proceedings of *11th IEEE ECBS*, 2004.

[12] H. Tianfield, "Multi-agent based autonomic architecture for network management," Proceedings of *IEEE INDIN*, 2003, pp. 462 - 469.

[13] E. Vassev and J. Paquet, "Towards an Autonomic Element Architecture for ASSL," Proceedings of *SEAMS '07*, 2007.

[14] H. Liu and M. Parashar, "Accord: A Programming Framework for Autonomic Applications," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 36, pp. 341-351, 2006.