Policy-based Adaptable Service Systems Architecture

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

Abstract

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

1. Introduction

Networked service systems are considered. *Services* are realized by the structural and behavioral arrangement of *service components*, which by their interworking provide a service in the role of a *service provider* to a *service 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 a service system which is able to adapt dynamically to changes in time and position related to users, nodes, status of capability and service performance measures, changed service requirements and policies.

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

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

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

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

In addition to this type of flexibility constituted by Actors that are able to play various EFSM-based

21st International Conference on Advanced Networking and Applications(AINA'07)
0-7695-2846-5/07 $20.00 © 2007 IEEE

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

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

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

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

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

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

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

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

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

2. Related work

Most of recent works related to policy-based adaptable service systems focuses on the aspects A) and B) as defined in Section 1. Examples are [5], [6], [7], which are Garlan et al.'s Rainbow architecture for self-adaptation, Samaan and Karmouch's autonomous policy-

based management framework and Narsi et al.'s learning techniques. A work that support the aspect C), PMAC by Agrawal et al. [2], uses Autonomic Computing Policy Language (ACPL) as a generic data model, which is analogous to our XML Equivalent Transformation (XET). However, this work has a weak focus on the aspects A) and B).

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

The architecture presented in this paper has the focus on all aspects A)-C). The service system framework permits the combination of both EFSM and the RM functionality. Considering the dynamic policy, i.e. the rule-based modification of the policy managing the service system (See Sections 4), the system's policy can be composed at run-time based on *evaluation criteria, reference inputs* and *feedbacks*. Unlike [6] and [7], the presented architecture evaluates and composes the best policy based on broad set of *evaluation criteria*, which can be history-based, prediction-based, or logic-based. *Income functions* are used as *reference inputs*, while the *feedbacks* are *system performance measures*.

3. Service System Framework

3.1 General component structure

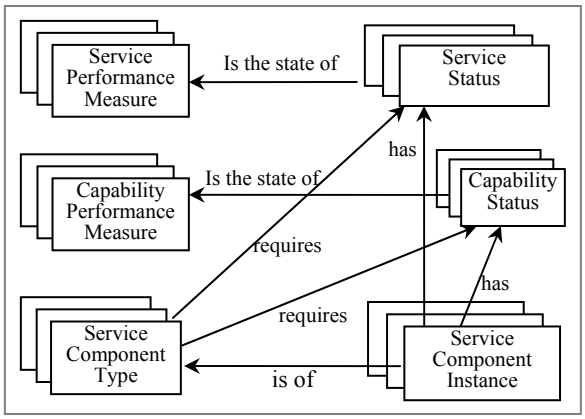

Figure 1 - Service System – General Component Structure

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

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

The following concepts are defined:

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

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

An EFSM type E is defined as:

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

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

computation on local data, communication initialization, database access, etc.

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

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

$$\mathcal{P} \equiv \{ \mathcal{X}, \mathcal{A} \}, \quad (2b)$$

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

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

3.2 Reasoning machine based functionality – application domain and integration

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

In addition to having RM functionality as a supplement to the EFSM-based service system, the RM functionality will also need EFSM support for the continuous updating of the system constraints, status data and reasoning conditions: $\mathcal{T}$, $\mathcal{E}$ and Σ, as well as the activation and deactivation of the reasoning machine based on the present value of Σ. The continuous updating of $\mathcal{T}$, $\mathcal{E}$ and Σ is done by EFSMs and in this case the $\mathcal{T}$, $\mathcal{E}$ and Σ is considered as common data for the EFSMs and the associated RM based functionality. A dedicated EFSM associated with the RM-based functionality denoted as E_Σ has the duty to inspect the reasoning condition and to activate and to deactivate the reasoning machine.

3.3 Reasoning procedure

The *Reasoning Procedure* is the procedure applied by the reasoning machine to select the rule to be applied. The *Reasoning procedure* is based on *Equivalent Transformation* (*ET*) [8], which solves a given problem by transforming it through repetitive application of (semantically) equivalent transformation rules.

ET consists of sets of *ET rules* and ET *clauses*. A problem must be formulated as a clause for transformation. An ET *clause* has the form:

$$\underbrace{\text{Head atom}}_{\textbf{Head}} \longleftarrow \underbrace{\text{Body atom}_1, \ldots \text{Body atom}_n}_{\textbf{Body}}$$

Head of a clause consists of an atom, or the *head atom*, which is a message containing a problem with unknown answer(s)/action(s). The problem in the head atom will be derived by *rules* until it eventually contains a list of suggested action(s).

An *ET rule* has the form:

$$\underbrace{\text{Head atom}}_{\textbf{Head}} \underbrace{\text{Condition atom}_1, \ldots}_{\textbf{Condition}} \longrightarrow \underbrace{\text{Body atom}_1, \ldots \text{Body atom}_n}_{\textbf{Body}}$$

It consists of a rule head and a rule body. A body atom of a clause matching the head atom of a rule can be transformed into the rule's bodies. A *rule of a policy* as defined in Section 3.1 is modeled by an *ET rule*

Intuitively, the reasoning procedure begins with a clause formulated by a message as follows:

$$\textbf{\textit{msg}(…)} \longleftarrow \textbf{\textit{msg}(…)} \quad (3)$$

The meaning of (3) is that the head *msg*(…) is true when the body *msg*(…) is true, which we don't need to prove. The goal of the reasoning procedure is to transform (3) until no body atom is left. Consider the following rule (4):

$$msg(\ldots), \mathrm{C} \xrightarrow{ET} B_1, B_2, \ldots B_n. \quad (4)$$

The rule (4) can transform the body atom msg(…) of (3) into B_1, B_2, … B_n; provided that the atom msg(…) match the head of (4) and the set of conditions C is not violated. Clause (3) will be transformed to (5) as follow:

$$msg(\ldots) \longleftarrow B_1, B_2, \ldots B_n. \quad (5)$$

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

4. Policy-based adaptation mechanism

4.1 System constraints, status data and reasoning conditions

The elements $\mathcal{T}$ and $\mathcal{E}$ of an RM as defined in Section 3, depend on the structuring and the nature of the reasoning functionality. They depend on which EFSMs that are related to the RM functionality and also the nature of the reasoning. A *reasoning cluster, which* is an independent unit with respect to reasoning, is a collection of EFSM-based service components with an associated *reasoning system* constituted by one or more *reasoning 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:

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

$$\mathcal{E} \equiv \{((\hat{E}_{lk}, \hat{S}_{I,lk}, \hat{C}_{I,lk}; l = [1, L_k]), k = [1, K]), (\hat{C}_{A,n}; n = [1, N_{Node}])\} \quad (7)$$

The function $Expr\{X_i; i = [1, I]\}$ in (6) symbolizes the set $\{X_i; i = [1, I]\}$ and also some set of logical functions based on the elements of the set. The system constraints $\mathcal{T}$ related to a reasoning cluster comprise the EFSM functionality sets of the EFSM-based service component types, required capability and service status, as well as the income functions for the reasoning cluster. The status data $\mathcal{E}$ defined in (7) is a set of the inherent capability and service status for all instances of EFSM-based service components in the reasoning cluster, as well as available capabilities of the nodes that potentially can contribute their capabilities for the EFSM based functionality of the reasoning cluster.

As stated in Section 1 service system adaptation also includes capability reconfiguration based on capability and service status requirements. *Capability configuration management* is the service systems initial capability configuration and reconfiguration. Capability configuration management goes beyond the boundaries of an individual reasoning clusters as well as an individual service system. This means that capability configuration management must be handled by a common distributed algorithm or by a centralized reasoning cluster.

The components constituting the *reasoning condition* Σ are the states and variables of $\hat{E}$, and the capability and service performance measures C and S as given in (8).

$$\Sigma \equiv Expr\ \{S, C, (\hat{E}_k, \hat{S}_{R,k}, \hat{C}_{R,k}; k = [1, K])\} \quad (8)$$

4.2 Policy-based adaptation using static policies

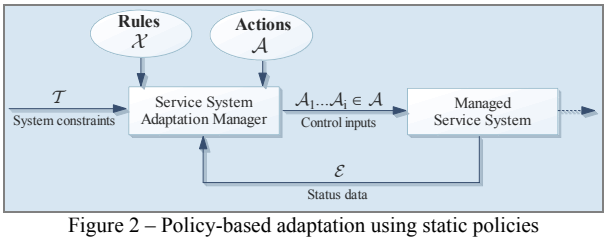

Figure 2 – Policy-based adaptation using static policies

The policy-based adaptation using static policies, as illustrated in Figure 2, manages the behavior of service systems based on system constraints $\mathcal{T}$. The managed service system give its status data $\mathcal{E}$ to the Service System Adaptation Manager $\mathcal{R}_1$, which is a reasoning machine. $\mathcal{R}_1$ gives a set of control inputs $\mathcal{A}_1 \ldots \mathcal{A}_i \subset \mathcal{A}$ (as already defined in Section 3.1) back to the managed service system. A policy system consists of static rules, which is unchangeable. Upon service systems enter a Σ_T, $\mathcal{R}_1$ is activated and try to lead the system back to a goal state Σ_G. $\mathcal{R}_1$ is de-activated when service systems enter Σ_G.

4.3 Policy-based adaptation using dynamic policies

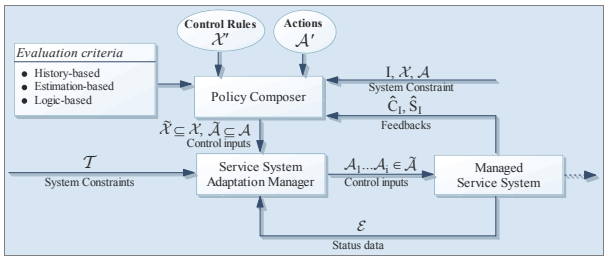

Figure 3 - Policy based adaptation based on dynamic policies

The use of static policies has some disadvantages. Firstly, it has only the priority mechanism to select a rule when two or more rules are applicable and it is hard to assign appropriate priorities to rules, especially when the rule space is large. Secondly, a set of static rules will not likely solve the problem under all set of different conditions that an adaptable service system must handle.

0-7695-2846-5/07 $20.00

A possible approach is to use dynamic policies as illustrated in Figure 3, and using two reasoning machines. In addition to the Service System Adaptation Manager ($\mathcal{R}_1$) a Policy Composer ($\mathcal{R}_2$) is used. A generic rule-based reasoning system with dynamic policy can be defined by (9a, 9b, 9c and 9d) as follows:

$$\mathcal{R}_1 \equiv \{ \mathcal{Q}, \mathcal{F}, \tilde{\mathcal{P}}, \mathcal{T}, \mathcal{E}, \Sigma \} \quad (9a)$$

$$\tilde{\mathcal{P}} \equiv \{ \tilde{\mathcal{X}}, \tilde{\mathcal{A}} \} \quad (9b)$$

$$\mathcal{R}_2 \equiv \{ \mathcal{Q}', \mathcal{F}, \mathcal{P}', \mathcal{T}', \mathcal{E}', \Sigma' \} \quad (9c)$$

$$\mathcal{P}' \equiv \{\mathcal{X}', \mathcal{A}' \} \quad (9d)$$

where $\mathcal{T}' = \{I, \mathcal{X}, \mathcal{A}\}$ and $\mathcal{E}' = \{\hat{C}_I, \hat{S}_I\}$. $\mathcal{Q}'$ is a set of messages between $\mathcal{R}_1$ and $\mathcal{R}_2$. $\mathcal{X}'$ 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 composer composes the system policy at runtime based on *evaluation criteria, reference inputs* and *feedbacks*. Evaluation criteria can be history-based, prediction-based and logic-based. Income functions are used as reference input, while the feedbacks are system performance measures.

The history-based evaluation method determines the consequences of the rules in the past. The prediction-based evaluation determines the consequences of rules in the future based on mathematical equations represented by $\mathcal{X}'$. The logic-based evaluation determines the consequences of rules based on logics such as fuzzy, business-level or user-defined logics represented by $\mathcal{X}'$.

5. Data Model

Data Model for the reasoning functionality is based on XML Equivalent Transformation language (*XET*), which is the XML representation of the ET as given in 3.3. XET represents status data, system constraints, and rules by using Extensible Markup Language (XML), Resource Definition Framework (RDF), XML expressions and XML rules.

XML and RDF: $\hat{C}_I, \hat{S}_I$ and $\hat{C}_A$ are expressed in the XML version of Common Information Model (CIM) denoted as xmlCIM [1]. RDF is used because CIM does not provide means for representing $\mathcal{Q}$ required by the developed framework [1]. $\hat{E}_I$ is modeled by the XML-based EFSM proposed by [9].

XML Expressions: XML expressions are used for the data representation of the system constraints $\mathcal{T}$. XML expressions [3] are ordinary XML elements with possible six disjoint variable types [1, 4]. XML expressions can represent implicit information by expressive XML variables. These variables can be specialized (or instantiated) into attributes names, element names, strings, zero or more attribute-value pair(s), one or more XML expression(s) and parts of XML expressions depending on their types (see [10]). An ordinary XML element (or a ground XML expression) is an XML expression without variables.

XET rule: An *XET* rule is the representation of an *ET* rule in XML. The structure for an XET rule is illustrated as follows.

```
<xet:Rule xet:name="..." xet:priority="..." xet:class="...">
  <xet:Meta>…</xet:Meta>
  <xet:Head>…</xet:Head>
  <xet:Condition>…</xet:Condition>
  <xet:Body>
    Body atom1, Body atom 2, …
  </xet:Body>
</xet:Rule>
```

A head, condition or body atom is represented by a fragment of XML. For example, the body `<xet:Body><a1/><a2/></xet:Body>` contains two body atoms, which are `<a1/>` and `<a2/>` respectively. The `xet:Meta` contains additional metadata for rules that will be used by control rules.

6. Application examples

6.1 The scenarios

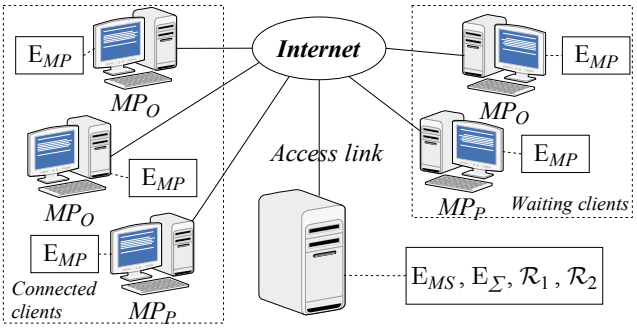

Figure 4 – A music video on-demand service system

Four scenarios handling the capability configuration management of a *music video on-demand service* are presented with the intention to *illustrate* the *use* of the proposed policy-based service system architecture, and the *potential advantages* of using dynamic policies. Scenario I and II use no policy. Scenario III uses static policies, while Scenario IV uses dynamic policies. The service system is constituted by one or more media servers (MS) streaming media files to media players (MP) (Figure 4). The numbers of MS used in Scenario I and II are fixed (one and two respectively), while the

number in Scenario III and IV can vary from one to two. An MP belongs to a *QoS_Class*. In the example two classes are applied: premium (MP_P) and ordinary (MP_O).

Three different streaming throughput bit-rates (X) are offered, 500Kbps, 800 Kbps and 1Mbps. MP_O connections are 500Kbps (X_O) while MP_P connections can be either 800Kbps or 1Mps (X_P).

The *capability performance measures* used are MS access link capacity (C_{AL}). The required and inherent MS capability status sets are defined as follows:

$$\hat{C}_R \equiv \{ C_{R,AL} \} \quad (10)$$

$$\hat{C}_I \equiv \{ C_{I,AL} \} \quad (11)$$

where $C_{R,AL}$ is the MS's required access link capacity, which is set to 100 Mbit/s.

The number of MPs that can use the service, in this example, is limited by the MS access link capacity. The service level agreements comprise *maximum waiting time*, *required streaming throughput*, *payment for the service* and *penalties for not satisfying the service*. The maximum waiting time for MP_P and MP_O are 60 seconds and infinite respectively. The required streaming throughput of MP_O and MP_P are X_P and X_O respectively. The payment for the service and the penalties for not satisfying the service are calculated by income and penalty functions that will be defined.

The resource management mechanisms used by the service provider is to *disconnect ordinary clients*, to *decrease the throughput of the premium clients* and to *change the number of media servers*.

When the required streaming throughput cannot be provided, an MP may have to wait until some connected MPs have finished using the service. This will result in money payback to the waiting MPs. An MP_O can be disconnected, while an MP_P may have to reduce the throughput. If a client is disconnected, the service provider pays a penalty. If the throughput is lowered, the price is lowered.

The *service performance measures* $\hat{S}_I$ consists of *the number of connected and waiting premium and ordinary clients* ($N_{Con,P}$, $N_{Con,O}$, $N_{Wait,P}$, $N_{Wait,O}$), *the number of disconnected MP_O* ($N_{Dis,O}$), *the number of MS* (N_{MS}), *inherent streaming throughput* (X_I), *the number of available nodes* (N_{Node}) and *the accumulated service time and waiting time of premium and ordinary clients* ($T_{Serv,P}$, $T_{Serv,O}$, $T_{Wait,P}$, $T_{Wait,O}$). These values are observed per a *monitoring interval* Δ.

A *cost unit* is the price paid by an ordinary customer for *one second streaming* of the rate 500 KBit/s. The income function for the service provider is m(QoS_Class, X_I) (cost units/second). The penalty function for waiting is p_{Wait}(QoS_Class) (cost units/second). The penalty function for disconnections is p_{Dis}(QoS_Class) (cost units/disconnection). The cost function for adding a new server is p_{Ser} (cost units per Node per sec). The total income function (m_T) during the monitoring interval Δ is defined as follows:

$$\begin{aligned} m_T = {} & m(MP_O, X_{I,O}) \times T_{Serv,O} + m(MP_P, X_{I,P}) \times T_{Serv,P} \\ & - p_{Wait}(MP_O) \times T_{Wait,O} - p_{Wait}(MP_P) \times T_{Wait,P} \\ & - p_{Dis}(MP_O) \times N_{Dis,O} - p_{Ser} \times (N_{MS}\text{-}) \times \Delta \end{aligned} \quad (12)$$

Policy-based adaptation is introduced to maximize the total income. The service system is realized as one reasoning cluster. As illustrated in Figure 4, E_{MS}, E_Σ, $\mathcal{R}_1$ and $\mathcal{R}_2$ are in the same node. E_{MS} is the *media server* type, E_{MP} is the *media player* type, $\mathcal{R}_1$ is the *service system adaptation manager* type and $\mathcal{R}_2$ is the *policy composer* type according to the definitions in Section 4.2 and 4.3. E_Σ, as defined in 3.2, is a delicate EFSM type for activate and de-activate $\mathcal{R}_1$ and $\mathcal{R}_2$. It is assumed that the initial capability configuration, as defined in Section 1, has taken place.

The nature of the *service system adaptation manager* as well as the need and nature of a *policy composer* depends on the difference in income and penalty for the different QoS classes, as well as the cost for introducing a new server. If the income and penalty for premium service class is relatively higher than for an ordinary class, it can be profitable to disconnect some MP_O and let some MP_P get the service instead.

The set of *actions* $\mathcal{A}$ applied for the *service system adaptation manger* applied in Scenarios III and IV consists of *Disconnect-Client* ($\mathcal{A}_D$), *Decrease-Bit-Rate* ($\mathcal{A}_B$), *Initialize-Server* ($\mathcal{A}_I$) and *Remove-Server* ($\mathcal{A}_R$). $\mathcal{A}$ can be defined by (13) as follows:

$$\mathcal{A} \equiv \{ \mathcal{A}_D, \mathcal{A}_B, \mathcal{A}_I, \mathcal{A}_R \} \quad (13)$$

$\mathcal{A}_D$ tells MS to disconnect a list of suggested MP_O. $\mathcal{A}_B$ tells MS to reduce throughput of a list of suggested MP_P for a certain time period. $\mathcal{A}_I$ tells MS to initiate a new MS, while $\mathcal{A}_R$ will remove an MS.

6.2 RM specification

6.2.1 Service system adaptation manager

The reasoning condition set for the service system adaptation manager is defined as follows:

$$\Sigma \equiv \{ \Sigma_{T1}, \Sigma_{G1} \} \quad (14)$$

where the reasoning activation condition (Σ_{T1}) is $N_{Wait,P}+N_{Wait,O} > 0$ and the reasoning goal condition (Σ_{G1}) is $N_{Wait,P}+N_{Wait,O} = 0$. The messages sent and received between MS and the service system adaptation manager is defined by $msg(\Sigma_T, \mathcal{A}_i)$.

The rule set $\mathcal{X}$ for the service system adaptation manger in Scenario III and IV is defined as follows:

$$\mathcal{X} \equiv \{ \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3, \mathcal{X}_4 \} \quad (15)$$

$\mathcal{X}_1$ suggests $\mathcal{A}_D$ for disconnecting a list of suggested MP_O. $\mathcal{X}_2$ suggests $\mathcal{A}_B$ for reducing throughput of a list of suggested MP_P. $\mathcal{X}_3$ suggests $\mathcal{A}_I$ for initiating a new MS, while $\mathcal{X}_4$ suggests $\mathcal{A}_R$ for removing an MS. $\mathcal{X}_1$, $\mathcal{X}_2$, $\mathcal{X}_3$, $\mathcal{X}_4$ can be further defined by as follows:

$$\mathcal{X}_1 \equiv msg(\Sigma_{T1}, \mathcal{A}_i) \{ p_{Wait}(MP_O) < p_{Wait}(MP_P) \} \longrightarrow \mathcal{A}_i \doteq \mathcal{A}_D. \quad (16)$$

When $p_{Wait}(MP_O) < p_{Wait}(MP_P)$, $\mathcal{X}_1$ will be executed. The suggested action $\mathcal{A}_i$ will be instantiated ($\doteq$) as $\mathcal{A}_D$, The number of chosen MP_O will be calculated as:

$$\left(\frac{N_{Wait,P} \times X_{P,1Mbps}}{X_O}\right)$$

$$\mathcal{X}_2 \equiv msg(\Sigma_{T1}, \mathcal{A}_i) \{ p_{Wait}(MP_O) > m(MP_P, X_{P,1Mbps}) - m(MP_P, X_{P,800Kbps}) \} \longrightarrow \mathcal{A}_i \doteq \mathcal{A}_B. \quad (17)$$

When $p_{Wait}(MP_O) > m(MP_P, X_{P,1Mbps}) - m(MP_P, X_{P,800Kbps})$, $\mathcal{X}_2$ will be executed. The suggested action $\mathcal{A}_i$ will be instantiated ($\doteq$) as $\mathcal{A}_B$. The number of MP_P to decrease bandwidth is calculated from the bandwidth that the waiting MP_O is needed divided by the difference between the possible bit-rate required by MP_P as:

$$\left(\frac{N_{Wait,O} \times X_O}{X_{P,1Mbps} - X_{P,800Kbps}}\right)$$

$$\mathcal{X}_3 \equiv msg(\Sigma_{T1}, \mathcal{A}_i) \left\{ \frac{X_P \times N_{WaitP} + X_O \times N_{WaitO}}{C_{R,AL}} > 0.1 \right\} \longrightarrow \mathcal{A}_i \doteq \mathcal{A}_I. \quad (18)$$

As given in (18), when the ratio of throughput required by all waiting MP (as shown in the rule's condition) and the capacity of an MS access link is more than 0.1, $\mathcal{X}_3$ will be executed and $\mathcal{A}_I$ will be suggested. A new MS can be initialized in a node having sufficient capabilities, which are $C_{R,AL}$ as defined in (10).

$$\mathcal{X}_4 \equiv msg(\Sigma_{T1}, \mathcal{A}_i) \left\{ \frac{X_P \times N_{WaitP} + X_O \times N_{WaitO}}{C_{R,AL}} < 0.1 \right\} \longrightarrow \mathcal{A}_i \doteq \mathcal{A}_R. \quad (19)$$

$\mathcal{X}_4$ suggests $\mathcal{A}_R$ when additional MS are not needed based on the ratio of the throughput required by all waiting MP and the access link capacity. If ratio is less than 0.1, $\mathcal{A}_R$ will be suggested.

6.2.2 Policy composer

In Scenario IV, reasoning conditions of the policy composer are defined as follow:

$$\Sigma' \equiv \{ \Sigma_{T2} \doteq \Sigma_{T1}, \Sigma_{G2} \doteq \Sigma_{G1} \} \quad (20)$$

The policy composer will always be activated whenever the service system adaptation manager is activated and will be de-activated whenever the service system adaptation manager is de-activated.

Upon entering Σ_{T2}, the service system adaptation manager sends a message $msg(\Sigma_{T2}, \mathcal{A}_i)$ to the policy composer. The set of messages $\mathcal{Q}'$ sent and received between them is defined as follow:

$$\mathcal{Q}' \equiv \{ msg(\Sigma_{T2}, \mathcal{A}_i) \} \quad (21)$$

The set of *actions* $\mathcal{A}$ applied for the *policy composer* in the Scenario IV can be defined as follows:

$$\mathcal{A}' \equiv \{\mathcal{A}_G(\mathcal{X}_i), \mathcal{A}_T(\mathcal{X}_i) \} \quad (22)$$

$\mathcal{A}_G(\mathcal{X}_i)$ is an action for the calculation of the *accumulated goodness score* of a rule $\mathcal{X}_i$. $\mathcal{A}_T(\mathcal{X}_i)$ is an action to suspend $\mathcal{X}_I$ for a certain time period. The *goodness score of a rule* ($Qo\mathcal{X}_i$) during the monitoring time interval T is calculated by the percentage of the increased or decreased total income (m_T). The algorithm to calculate $Qo\mathcal{X}_i$ is as follows:

$$Qo\mathcal{X}_i = Qo\mathcal{X}_i + \frac{m_{T,t} - m_{T,t-1}}{m_{T,t}} \times 100 \quad (23)$$

where $m_{T,t}$ and $m_{T,t-1}$ are the total income during the current and previous monitoring interval respectively.

The rule set $\mathcal{X}'$ of the policy compose*r* applied in Scenario IV is defined as follow:

$$\mathcal{X}' \equiv \{ \mathcal{X}'_1, \mathcal{X}'_2 \} \quad (24)$$

where $\mathcal{X}'_1$, $\mathcal{X}'_2$ (See Sec. 6.2.3) can be defined by (25) and (26) as follows:

$$\mathcal{X}'_1 \equiv msg(\Sigma_{T2}, \mathcal{A}_i) \{\mathcal{X}^{t-1} = \mathcal{X}_i \} \longrightarrow \mathcal{A}_i \doteq \mathcal{A}_G(\mathcal{X}_i). \quad (25)$$

When the policy composer finds that a rule $\mathcal{X}_i$ has been executed during the last interval t-1 ($\mathcal{X}^{t-1} = \mathcal{X}_i$), the policy composer executes $\mathcal{X}'_1$. The suggested action $\mathcal{A}_i$ will be instantiated as $\mathcal{A}_G(\mathcal{X}_i)$.

$$\mathcal{X}'_2 \equiv msg(\Sigma_{T2}, \mathcal{A}_i) \{ Qo\mathcal{X}_i < 0 \} \longrightarrow \mathcal{A}_i \doteq \mathcal{A}_T(\mathcal{X}_i). \quad (26)$$

When the goodness score of a rule is less than zero, the policy composer executes $\mathcal{X}'_2$. The suggested action $\mathcal{A}_i$ will be instantiated as $\mathcal{A}_T(\mathcal{X}_i)$.

6.3 Results

The MP arrivals are modeled as a Poisson process with parameter λ_{QoS_Class}. The duration of streaming connections (d_{QoS_Class}) is constant. The quantity $\rho = ((\lambda_O \times d_O \times X_O)+(\lambda_P \times d_P \times X_P))/C_{I,AL}$ is the sum of traffic offered to MS access links. Note that ρ can be larger than the number of MS access links. The duration of streaming connections are set to 4 minutes, while the monitoring interval Δ is set to 1 minute.

Table 1 – Income and penalty functions in *cost units*

	MP_O	MP_P X_I = 800Kbps	MP_P X_I = 1Mbps
m(QoS_Class, X_I) (per second)	$\frac{1}{60}$	$\frac{2}{60}$	$\frac{1.75}{60}$
p_{Wait}(QoS_Class) (per second)	$\frac{1}{3}$	$\frac{5}{3}$	$\frac{5}{3}$
p_{Dis}(QoS_Class) (per disconnection)	$\frac{5}{3}$	-	-

All Scenarios were tested for 500 minutes with two ρ values: 0.42 and 0.84. The MP_P arrival intensity is 15% of the total arrival intensity. The income and penalty functions in cost units are given in Table 1. The cost for using an extra MS is 417 cost units per Node per second.

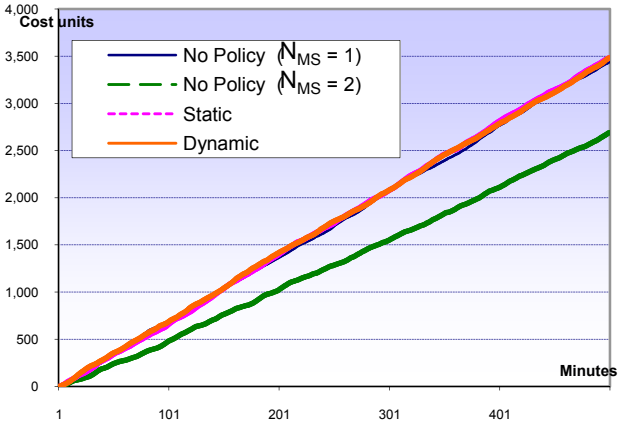

Figure 5 – the accumulated total income of all scenarios (ρ = 0.42)

Figure 5 illustrates the accumulated total income for all scenarios when ρ = 0.42. The values of accumulated total income of Scenario I (No Policy, N_{MS} = 1), Scenario III (static policies) and Scenario IV (dynamic polices) are identical, while the accumulated total income in the Scenario II (No Policy, N_{MS} = 2) is lower.

The low traffic implies that no rule is applied in Scenario III and IV. This made the outcome of Scenario I, III and IV identical. On the other hand, the cost of an extra server, which is not necessary for such arrival intensity, decreased the total income of the system.

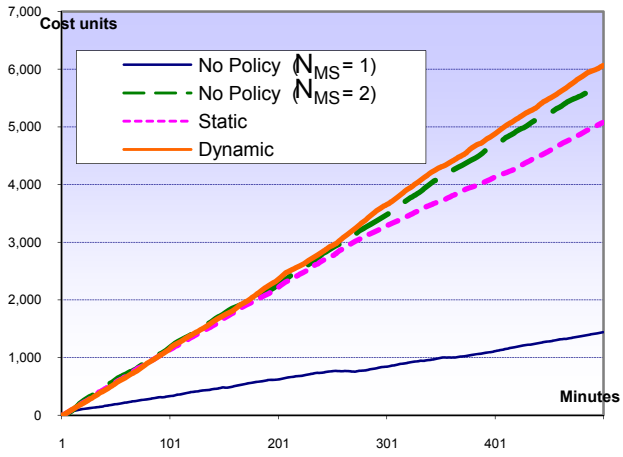

Figure 6 – the accumulated total income of all scenarios (ρ = 0.84)

Figure 6 Illustrates the accumulated total income of all scenarios when ρ = 0.84. As a result, the accumulated total income of Scenario I was much lower than the others as MP must wait to get the service. Scenario II using two MS and no policy gives a very good result. There was no MP waiting during the test and no penalty was paid.

Scenario III and IV were started out with one MS. The number of MS were being increased or decreased by $\mathcal{X}_3$ and $\mathcal{X}_4$. In addition, $\mathcal{X}_1$ and $\mathcal{X}_2$ also manage the link capacity of MS by disconnecting MP_O or decrease the MP_P throughput. The accumulated income in Scenario IV was higher than for Scenario III.

It is observable on both Scenario III and IV that the use of $\mathcal{X}_4$, which will remove an MS, apparently reduced the system's accumulated total income. Having two servers all the time seems to be better for the high traffic, provided that the extra server cost is not too high. The dynamic policies suspend $\mathcal{X}_4$ for 50 minutes and thus lengthen the time where two MSs are in operation. In Scenario IV, $\mathcal{X}_4$ was executed 26 times comparing to 35 times in Scenario III.

For the present scenarios none of the non-policy scenarios (I and II) gave good results for both the low and high traffic case. The policy-based scenarios seem to be more suitable with respect to good results over a variety of system load conditions. The accumulated total income in Scenario III and IV also have the potential to be improved by changing the XML-based policies.

0-7695-2846-5/07 $20.00

7. Conclusion

An architecture for policy-based adaptable service systems based on the combination of Reasoning Machines (RM) and Extended Finite State Machines (EFSMs) has been presented. The architecture comprises *service system framework, adaptation mechanisms* and *data model*. Policies have been introduced with the intension to increase flexibility in the adaptable service system specification and execution.

The service components constituting the *service system* are modeled by some combination of EFSM type and RM type. The RM, which is controlled by a specific purpose EFSM denoted as $EFSM_{\Sigma}$, is an independent component. The reasoning procedure applied by the RM is based on Equivalent Transform (ET).

The *Adaptation mechanism* uses policies to control service systems when it is entering a reasoning condition. The use of policy can be of two types: *static* or *dynamic*. In the static case the reasoning system *constituted by a service system adaptation manager* determines a list of suggested actions that will control the behavior of the service system. In the dynamic case an additional RM, denoted as the *policy composer*, is added. The *policy compose*r is able to compose policy on-the-fly, and has the ability to estimate or evaluate the consequences of the rules of a policy based on their goodness scores.

The Data Model based on XML Equivalent Transformation (XET) is used to express system constraint, system status, reasoning conditions, rules and control rules. The XML-based specifications are readily executable by XET-based RM. This also represents a flexibility feature of the proposed architecture.

Four scenarios handling the capability configuration management of a music video on-demand service are presented with the intention to illustrate the use of the proposed architecture and the potential advantage of using dynamic policies. Scenario I and II use no policies. Scenario III uses static policies, while Scenario IV uses dynamic policies. There are situations where the use of no policy can be superior or equal to the use of policies. The selected system parameters can represent an optimal dimensioning. However, the same set of system parameters will likely not be optimal for other system traffic load cases. For the presented scenarios the use of no policy and one server is a good solution in the low traffic case, while the use of no policy and two servers is a good solution in the high traffic case.

In the given scenarios, the service system operated under static policies give a relatively high income in both low and high traffic. The service system operated under dynamic policies, however, has a performance which is superior or equal to other scenarios in both the low and the high traffic case. In addition to having the potential for providing optimal solutions covering dynamic traffic situations, the proposed architecture also is a flexible tool for the experimentation with alternative policies with respect to optimization.

References

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

[2] D. Agrawal, K.-W. Lee, and J. Lobo, "Policy-Based Management of Networked Computing Systems," *IEEE Communications Magazine,* vol. 43, pp. 69-75, 2005.

[3] Y. Diao, J. L. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung, "A Control Theory Foundation for Self-Managing Computing Systems," *IEEE Journal on Selected Areas in Communications,* vol. 23, pp. 2213-2222, 2005.

[4] P. Supadulchai and F. A. Aagesen, "A Framework for Dynamic Service Composition," in *First International IEEE Workshop on Autonomic Communications and Computing (ACC 2005)*, Taormina, Italy, 2005.

[5] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-Based Self-Adaptation with Reusable Instrastructure," *Computer,* vol. 37, pp. 46-54, Oct 2004 2004.

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

[7] R. Nasri, Z. Altman, and H. Dubreil, "Autonomic Mobile Network Management Techniques for Self-Parameterisation and Auto-regulation," in *Smartnet 2006*, Paris, 2006.

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

[9] S. Jiang and F. A. Aagesen, "XML-based Dynamic Service Behaviour Representation," in *NIK'2003*, Oslo, Norway, 2003.

[10] P. Supadulchai, "List of XML Variables, http://tapas.item.ntnu.no/wiki/index.php/XML_Variables," 2007.

0-7695-2846-5/07 $20.00 © 2007 IEEE