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 user*. Service components are executed as software components in *nodes*, which are physical processing units such as servers, routers, switches and user terminals.

An adaptable service system is here defined as a service system which is able to adapt dynamically to changes in time and position related to users, nodes, capabilities, system performance, changed service requirements and policies. In this context, *capability* is defined as an inherent physical *property* of a node, which is used as a basis to implement services. Capabilities can be classified into *resources*, *functions* and *data*. Examples are CPU, memory, transmission capacity of connected transmission links, available special hardware, and available programs and data.

The software mechanisms used for implementing the functionality of the service components of adaptable service systems must be flexible and powerful. Service components based on the classical EFSM (Extended Finite State Machine) approach can be flexibly executed by using generic EFSM executing software components that are able to download and execute different EFSM-based specifications [1].

In addition to this type of flexibility the *EFSM-based* functionality can be supplemented by *reasoning-machine* (RM) based functionality, which makes policy-based specification and operation possible. *"Policies represent externalized logic that can determine the behavior of the managed systems"* [2]. In this paper *a policy is technically defined as a set of rules with related actions.* A *policy system* is a set of policies, and an *RM-based functionality is using a policy system to manage the behavior of a target system, which can be* another policy system. A *static policy system* has a non changeable set of rules and actions, while a *dynamic policy* system has a changeable set of rules and actions.

Policy-based software has a specification style, which is expressive and flexible. Software functionality based on policy-based specifications, however, also needs to be appropriately specified and validated. The validation aspect is outside the scope of this paper.

The main contribution of this paper is the presentation of a generic service framework for adaptable service systems that combines the use of EFSM-based and RM-based service components. In this context the reasoning machines can be used

a) as ordinary procedural services for EFSM-based service components
b) for instantiation and re-instantiation (i.e. after movement) of EFSM-based service components according to the availability and need of capabilities
c) to adapt the behavior of and capabilities allocated to instantiated EFSM-based service components in the nodes where they are instantiated

This paper has focus on issue c), but the framework presented can be used for a) and b) also. In general, adaptation needs appropriate mechanisms to guarantee the wanted results. For autonomous adaptation stable feedback loops [3], which control the performance, are needed. As the capabilities are limited, the access to the system must be controlled, and there must also be priority mechanisms that give priority to users which are willing to pay more and/or are in a higher need in situations with lack of capabilities.

The issues of policy-supported adaptable service system architecture are in this paper classified into 3 main aspects: A) Service system framework, B) Adaptation mechanism and C) Data model. *Service system framework* comprises abstraction, concepts and models. *Adaptation mechanism* concerns the use of the appropriate policies to control the service system when it is entering a state where RM functionality is needed. *Data model* concerns the data representation of the service system framework and adaptation mechanisms.

This paper comprises the aspects A) and B) only. For details about the data model, which is based on XML Equivalent Transformation language (*XET*), Common Information Model (CIM) and Resource Definition Framework (RDF), the reader is referred to [1] and [4]. The remaining part of this paper is structured as follows. Section 2 discusses related work. Section 3 presents the service system framework. Section 4 presents policy-based adaptation mechanism. Section 5 presents the models and results for example application cases related to capability management of a music video on-demand service. Section 6 gives summary and conclusions.

2 Related Work

Most of recent works related to policy-based adaptable service systems focus on the aspects A) and B) as defined in Section 1. Examples are [2, 5-10]. The aspect C) is supported by XML-based language in [2, 10], which is analogous to our used XML

Equivalent Transformation (XET). However, [2] has a weak focus on the aspects A) and B), while [10] has a weak focus on A).

Considering the nature of the policies, [5] is preliminary aimed at static policies, while [6-10] are both using static and dynamic policies. Excluding [8], systems capable of dynamic policies [6-7, 9, 10] are based on proper feedbacks. The feedback loops in [5, 7, 9, 10] are used to evaluate the service system rather than policies. The loop in [6] evaluates policies. However, the evaluation is based on complex mathematical equations and not by additional policy sets.

The adaptation mechanisms presented in this paper can use static as well as dynamic policies. Considering the dynamic policy, the rule-based modification of the policy managing the service system can be composed at run-time.

The use of dynamic policies in [9, 10] as well as in this paper also aims at being a flexible tool for the experimentation with alternative policies with respect to optimization.

3 Service System Framework

The concept *capability* was defined in Section 1. *Capability performance measures* are the concepts used for the performance modeling, dimensioning, analyzing, monitoring and management of capabilities. Capability performance measures comprise capability *capacity*, capability *state* and capability *Quality of Service* (QoS) measures (e.g. traffic and availability measures). *Service performance measures* are performance measures related to the service provided to the service user (e.g. QoS measures) as well as service system state measures.

An executing service system consists of executing service components which are *instances* of *service component type*s. The functionality types are *EFSM types* and *RM types.* The basic functionality of the service components, however, are based on EFSMs supported and/or controlled by RMs. EFSM components will have requirements with respect to *capability* and *service performance* to be able to perform their intended functionality (Fig. 1). These requirements are denoted as required capability and service performance. The capability and service performance of an executing service system are denoted as inherent capability and service performance.

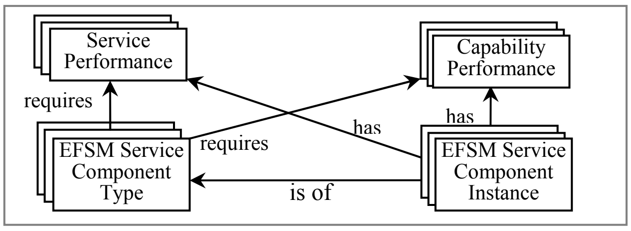


Fig. 1. EFSM part of Service System – Concept Structure

Capability management (CM) is an important function within an adaptable service system and comprises: 1) service system capability initialization, 2) capability allocation adaptation and 3) capability re-initialization. *Service system capability initialization* is the allocation of the capabilities for the service components to be distributed and instantiated. Capabilities are allocated according to the system performance requirements of the EFSM components of a service system. *Capability allocation adaptation* is the monitoring of the performance of the executing service system and the reallocation of capabilities within the executing service systems. In situations when the instantiated service systems are unable to adapt satisfactory, capability management can initiate a *service system capability re-initialization* for a re-distribution and re-instantiation of the service system.

As a basis for the optimal adaptation, *service level agreements* (SLA) are needed between the *service users* and the *service provider*. The service provider view of this service level agreement can in this context be considered as a part of executing service components. A number of QoS levels can exist. The agreement can contain elements such as: agreed QoS levels, required capabilities, required system performance, payment for the service in case of agreed QoS level and payments for the service in case of reduced QoS level. A service level agreement class (SLA class) defines provided service user functionalities as well as agreed QoS parameter and cost values for a group of service users with different degree of satisfactions and cost.

In the following a formalized service framework model is presented. The following concepts are defined:

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 ($\equiv$) as follows:

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

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

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

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

Concerning $\mathcal{R}$ and $\mathcal{P}$, $\mathcal{Q}$ is the set of messages, $\mathcal{F}$ is a generic *reasoning procedure*, $\mathcal{P}$ is a policy system which consists of a set of rules $\mathcal{X}$ and a set of actions $\mathcal{A}$, $\mathcal{T}$

is a set of *system constraints* and $\mathcal{E}$ is a set of *performance data*. The *reasoning procedure* is the procedure applied by RM to select the appropriate actions. The performance data represents the inherent performance of the targeted system. The system constraints represent the variables of the system and the defined constraints and relationships between variables. The policy rules are based on the variables of the constraints. Σ is a set of reasoning conditions defined by *trigger conditions* Σ_T, *and goal conditions* Σ_G. RM functionality is activated when a Σ_T is detected until a Σ_G is reached. When a trigger condition is true, the reasoning procedure transforms $\mathcal{Q}_i$ to $\mathcal{Q}_j$ by using $\mathcal{P}$ to match the system constraints $\mathcal{T}$ against the *performance data* $\mathcal{E}$ and a set of suggest actions $\{\mathcal{A}_i, \mathcal{A}_j, \mathcal{A}_k\ldots\} \subseteq \mathcal{A}$. These actions may also set the next state and values of the variables of EFSM-based service component instances. The *reasoning procedure* is based on *Equivalent Transformation* (*ET*) [11], which solves a given problem by transforming it through repetitive application of (semantically) equivalent transformation rules.

The RM functionality will need EFSM support for the continuous updating of $\mathcal{T}$, $\mathcal{E}$ and Σ, and for the activation and deactivation of the reasoning machines. This is done by EFSMs, and in this case $\mathcal{T}$, $\mathcal{E}$ and Σ are considered as common data for the EFSMs and the associated RM-based functionality. A dedicated EFSM E_Σ has the duty to inspect the reasoning condition and to activate and to deactivate the reasoning machine.

4 Policy-Based Adaptation Mechanism

4.1 System constraints, performance data and reasoning conditions

The elements $\mathcal{T}$ and $\mathcal{E}$ of an RM as defined in Section 3 depend on the structuring and the nature of the reasoning functionality. A *reasoning cluster, which* is an independent unit with respect to reasoning, is a collection of EFSM-based service components with an associated *reasoning system* constituted by one or more *reasoning machine*s. A reasoning cluster has a set of associated income functions I. The elements $\mathcal{T}$ and $\mathcal{E}$ of a reasoning cluster with available capabilities from N_{Node} nodes, consisting of K EFSM-based service component types and L_k instances of an EFSM-based service type k are defined as follows:

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

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

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).

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

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.

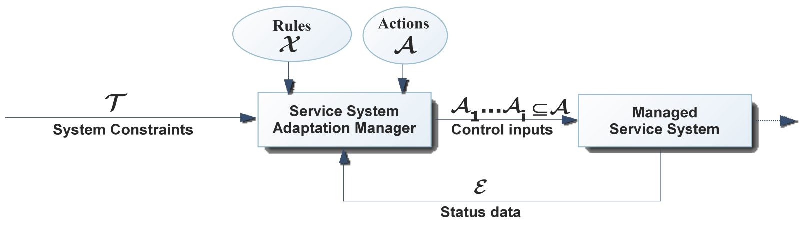
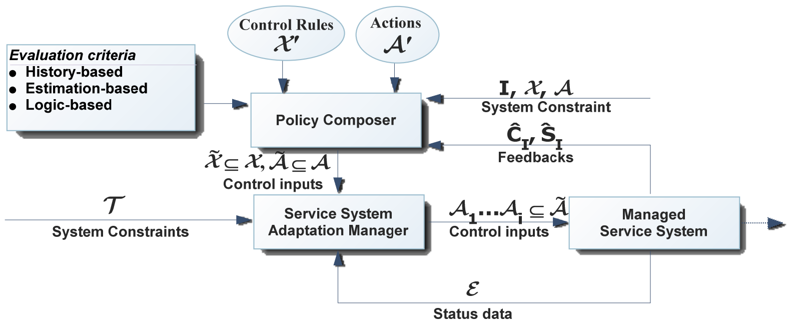


Fig. 2. Policy-based adaptation using static policies

4.3 Policy-based adaptation using dynamic policies



Evaluation criteria
History-based
Estimation-based
Logic-based
Control Rules $\mathcal{X}'$
Actions $\mathcal{A}'$
Policy Composer
$\mathbf{I}, \mathcal{X}, \mathcal{A}$
System Constraint
$\hat{\mathbf{C}}_\mathbf{I}, \hat{\mathbf{S}}_\mathbf{I}$
Feedbacks
$\tilde{\mathcal{X}} \subseteq \mathcal{X}, \tilde{\mathcal{A}} \subseteq \mathcal{A}$
Control inputs
$\mathcal{T}$
System Constraints
Service System Adaptation Manager
$\mathcal{A}_1 \ldots \mathcal{A}_i \subseteq \tilde{\mathcal{A}}$
Control inputs
Managed Service System
$\mathcal{E}$
Status data

Fig. 3. Policy based adaptation based on dynamic policies

The adaptation mechanism using dynamic policies is illustrated in Fig. 3. In addition to the Service System Adaptation Manager ($\mathcal{R}_1$) a Policy Evaluator ($\mathcal{R}_2$) is used.

A generic rule-based reasoning system with dynamic policy can be defined by (6a, 6b, 6c and 6d) as follows:

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

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

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

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

where $\mathcal{T}' = \{I, \mathcal{X}, \mathcal{A}\}$ and $\mathcal{E}' = \{\hat{C}_I, \hat{S}_I\}$. $\mathcal{Q}'$ is a set of messages between $\hat{\mathcal{R}}_1$ and $\hat{\mathcal{R}}_2$. $\mathcal{X}'$ is a set of control rules that can re-order the priority of the rules, activate and 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* (T_L), which is the time needed by the system to properly evaluate the rules. The *learning time* T_L depends on the service performance measures used by the evaluation algorithm. However, there is no unique and easy way to define T_L.

5 Application Examples

5.1 The application cases

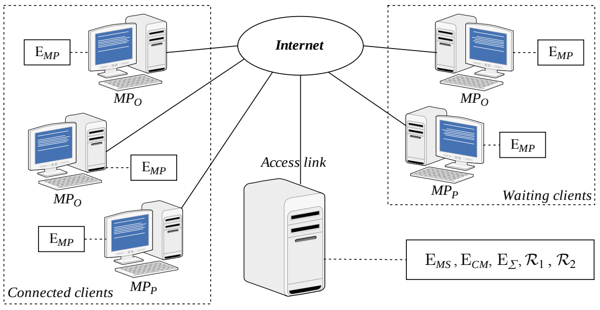


Fig. 4. A music video on-demand service system; E_{MS}: Media server type, E_{MP}: Media player type, E_{CM}: Capability manager type, E_{Σ}: Dedicated EFSM type for controlling the reasoning mechanism, $\mathcal{R}_1$: Service system adaptation manager type, $\mathcal{R}_2$: Policy evaluator type.

Five application cases (Case I-V) for a simple service system handling the capability management for a music video on-demand service is presented. The intention is to illustrate the use of the proposed policy-based service system architecture, and the potential advantages of using dynamic policies. The Cases I - III use no policy, Case IV uses static policies, while Case V uses dynamic policies. The service system is constituted by one or more media servers (MS) streaming media files to media players (MP) (Fig. 4). The numbers of MS used in Case I, II and III are fixed (one, two and three respectively), while the number in Case IV and V can vary from one to three.

The basic EFSM types constituting the capability management system are media server handler (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:

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

The reasoning machine supported capability manager will try to maximize the total income. The service system is realized as one reasoning cluster as illustrated in Fig. 4. The nature of the *service system adaptation manager* as well as the need and nature of a *policy evaluator* depends on the difference in income and penalty for the different SLA classes, as well as the cost for introducing a new server. If the income and penalty for premium service class is relatively higher than for an ordinary class, it can be profitable to disconnect some MP_O and let some MP_P get the service instead.

The specification of the behavior of the *service system adaptation manager* used for the Cases IV and V, and the *policy evaluator* applied for the Case V is given in Appendix.

5.2 Results

Table 1. Income and penalty functions

	MP_O	MP_P (X_I = 600Kbps)	MP_P (X_I = 1Mbps)
m(SLA_Class, X_I) / s	1	1.875	2
p_{Wait}(SLA_Class) / s	5	10	10
p_{Dis}(SLA_Class) / disconnection	10	-	-

The MP arrivals are modeled as a Poisson process with parameter λ_{SLA_Class}. The duration of streaming connections d_{SLA_Class} is constant. The quantity $\rho = ((\lambda_O\times d_O\times X_{O)}) + (\lambda_P\times d_P\times X_{P)}))/C_{I,AL}$ is the traffic per an MS access link. Intuitively, the system with $\rho \leq 1$ needs at least one server while the system with $1 \leq \rho \leq 2$ needs at least two servers and so on. The MP_P arrival intensity is 15% of the total arrival intensity. The duration of streaming connections are set to 10 minutes, while the monitoring interval Δ is set to 1 minute. MPs stop waiting after 10 minutes. The income and penalty functions in units are given in Table 1.The cost for using an extra MS is 833 units/s per Node.

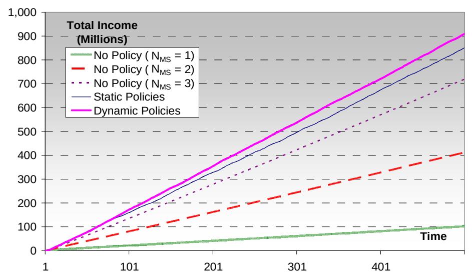


Fig. 5. Accumulated total income for ρ = 3.45

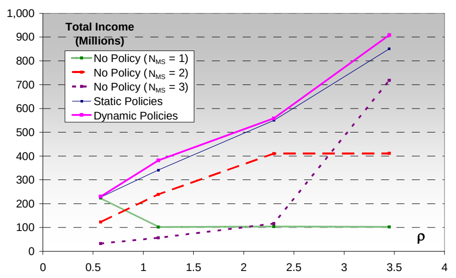


Fig. 6. Accumulated income at 500th ms

Fig. 5 illustrates the accumulated total income when ρ = 3.45. The value 3.45 is chosen to compare the no-policy scenarios with N_{MS} = 1, 2, or 3 and as well as the static and dynamic policy scenarios. The accumulated total incomes of cases with no policy are relatively lower than those with policies.

Fig. 6 illustrates the values of accumulated total income at the 500th minute for the ρ values: 0.56, 1,2, 2.3 and 3.5. The systems with no policy produce good results with a certain load region. The systems operated under policies produced higher accumulated total income independent of load region. Dynamic policies give relatively better result. These cases also have the potential improvement by changing the policies.

Fig. 7 shows the system behavior for Case IV and V when the traffic is being increased or decreased (the value of ρ varies as a function of time). The time with ρ at a fixed level is denoted as the ρ *period*. The dotted line shows the variation of ρ, which can take the values 0.5, 1, 1.5 and 2 times of ρ = 1.44. The ρ period, which is $10\times d_{\mathrm{SLA\_Class}}$, provides much time for the system for learning the consequences of the rules being applied. Case V gives a better result.

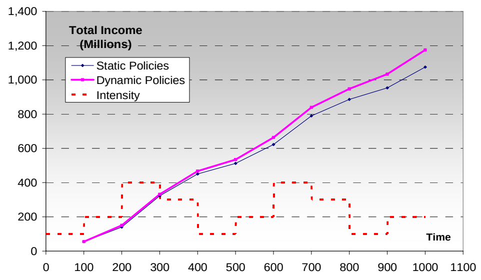


Fig. 7. Accumulated total income

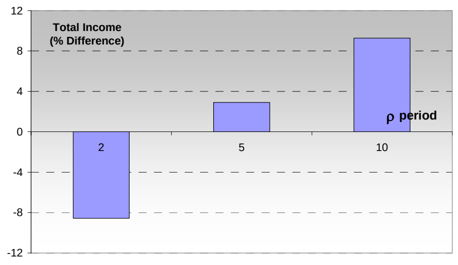


Fig. 8. Comparison of Case IV&V

Fig. 8 shows a comparison between Case IV and V for different ρ periods. The figure shows the difference between the values of accumulated total income after 500 minutes. When the ρ period is small, Case IV may give better result because the system need more learning time (T_L). The T_L value falls between $2\times$ and $5\times d_{\mathrm{SLA\_Class}}$.

The use of $\mathcal{X}_3$, $\mathcal{X}_4$ (see Appendix), which will add or remove an MS, affects the system's accumulated total income. Having more MS all the time is better for high traffic while having few MS all the time is better for low traffic. The policy evaluator learned this by observing the consequences of $\mathcal{X}_3$ and $\mathcal{X}_4$. The ability to learn can also be improved by appropriately selecting service performance measures and algorithms.

6 Conclusion

An architecture for policy-based adaptable service systems, based on the combination of Reasoning Machines (RMs) and Extended Finite State Machines (EFSMs) has been presented. Policies have been introduced with the intension to increase flexibility in the system specification and execution.

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

Five application cases handling the capability management of a music video on-demand service are presented. The intention is to illustrate the use of the proposed architecture and demonstrate the potential advantage of using dynamic policies. Case I, II and III use no policies. Case IV uses static policies, while Case V uses dynamic policies. Only capability allocation adaptation is considered. There are situations where the use of no policy can be superior or equal to the use of policies. The selected system parameters can represent an optimal dimensioning. However, the same set of system parameters will likely not be optimal for other system traffic load cases. The service system operated under static policies give a relatively high income in both low and high traffic. The service system operated under dynamic policies, however, has a performance which is superior or equal to other application cases. Nevertheless, the service system operated under dynamic policies needs a certain period of time denoted as *learning time* to learn the consequences of policies in order to provide superior performance.

The proposed architecture is also a flexible tool for the experimentation with alternative policies with respect to optimization.

References

1. F. A. Aagesen, P. Supadulchai, C. Anutariya, and M. M. Shiaa, "Configuration Management for an Adaptable Service System," in IFIP International Conference on Metropolitan Area Networks, Architecture, Protocols, Control, and Management, Ho Chi Minh City, Viet Nam, 2005.
2. D. Agrawal, K.-W. Lee, and J. Lobo, "Policy-Based Management of Networked Computing Systems," *IEEE Communications Magazine,* vol. 43, pp. 69-75, 2005.
3. Y. Diao, J. L. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung, "A Control Theory Foundation for Self-Managing Computing Systems," *IEEE Journal on Selected Areas in Communications,* vol. 23, pp. 2213-2222, 2005.
4. P. Supadulchai and F. A. Aagesen, "A Framework for Dynamic Service Composition," in *First International IEEE Workshop on Autonomic Communications and Computing (ACC 2005)*, Taormina, Italy, 2005.
5. D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-Based Self-Adaptation with Reusable Instrastructure," *Computer,* vol. 37, pp. 46-54, Oct 2004 2004.
6. N. Samaan and A. Karmouch, "An Automated Policy-Based Management Framework for Differentiated Communication Systems," *IEEE Journal on Selected Areas in Communications,* vol. 23, pp. 2236-2247, 2005.
7. R. Nasri, Z. Altman, and H. Dubreil, "Autonomic Mobile Network Management Techniques for Self-Parameterisation and Auto-regulation," in *Smartnet 2006*, Paris, 2006.
8. Y. Kanada, "Dynamically Extensible Policy Server and Agent," in *Proceedings of the 3rd Int'l Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, 2002.
9. H. Chan and T. Kwok, "A Policy-based Management System with Automatic Policy Selection and Creation Capabilities using a Singular Value Decomposition Technique," in *Proceedings of the 7th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06)*, 2006.

10. R. J. Anthony, "A Policy-Definition Language and Prototype Implementation Library for Policy-based Autonomic Systems," in *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, 2006.
11. K. Akama, T. Shimitsu, and E. Miyamoto, "Solving Problems by Equivalent Transformation of Declarative Programs," *Journal of the Japanese Society of Artificial Intelligence,* vol. 13, pp. 944-952, 1998.

Appendix. Reasoning Machine Specifications

1. Service system adaptation manager (Case IV and V)

The set of actions $\mathcal{A}$ applied by the service system adaptation manger is:

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

$\mathcal{A}_D$ *(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:

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

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.