# An Autonomic Framework for Service Configuration

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

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

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

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

## I. INTRODUCTION

Networked service systems are considered. Services are realized by service components, which by their inter-working provide a service in the role of a service provider to service users [1]. Service components are executed as software components in nodes, which are physical processing units such as servers, routers, switches, PCs and mobile phones. A service framework is here defined as the overall structural and behavior framework for the specification and execution of services. *Service configuration* comprises capability configuration, capability allocation, service deployment and instantiation, system performance diagnosis, fault diagnosis and service adaptation. *A capability* is an inherent property of a node required as a basis to implement services [1]. Capabilities can be classified into resources, functions and data. Examples are CPU, memory, transmission capacity of connected transmission links, available special hardware, and available programs and data.

The service configuration is done with respect to required capabilities and capability performances as well as required service performances. In this paper, we focus on service configuration of *adaptable service systems*, here defined as a service system that can adapt by itself related to changes by users, nodes, capabilities, system performances and service functionalities.

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

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

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

## II. PROPERTIES OF AUTONOMIC ELEMENTS

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

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

AEs have the following *shared* properties:

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

## III. GOAL-BASED POLICY ONTOLOGY

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

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

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

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

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

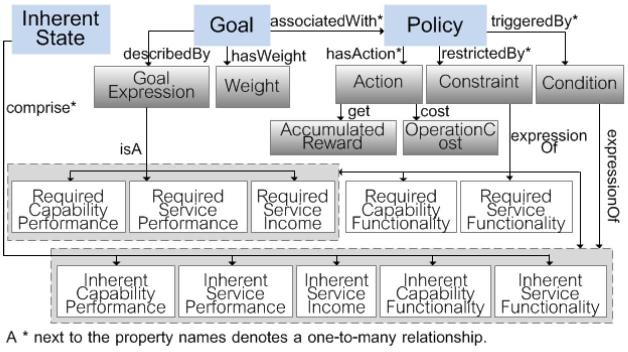
Inherent State
Goal
associatedWith*
Policy
triggeredBy*
describedBy
hasWeight
hasAction*
restrictedBy*
Goal Expression
Weight
Action
Constraint
Condition
comprise*
get
cost
Accumulated Reward
OperationCost
expression Of
expressionOf
isA
Required Capability Performance
Required Service Performance
Required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

Figure 1. Goal-based Policy Ontology.

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

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

TABLE I. THE CAPABILITY, SERVICE AND INCOME CONCEPT NOTATION

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

## IV. AUTONOMIC ELEMENT MODEL

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

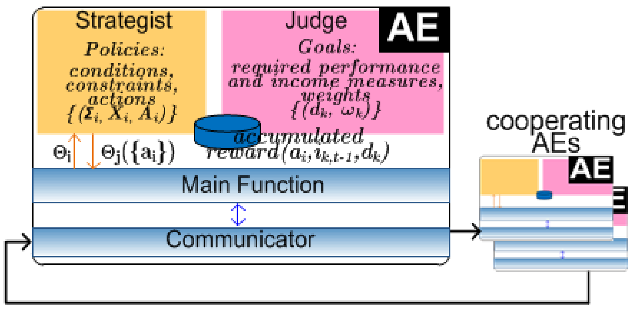

Figure 2. Autonomic Element Model.

## A. *Main Function*

*Main Function* coordinates the functionality of an AE. An AE has some general behavior which is common for all AEs, and some behavior depending on the specific role of the AE (see Section V). In general, an AE will have requirements with respect to capabilities and capability functionalities and performances. The specific need depends on the specific functionality of the AE. The *Main Function* behavior is based on an *Extended Finite State Machine (EFSM)* model E defined (≡) as*:*

$$E \equiv \{ S_M, S_I, S_S, V, M, O, Q, F_S, F_O, F_V \} \qquad (1)$$

where $S_M$ is a set of all states, $S_I$ is an initial state and $S_S$ is a set of *stable* states. V is a set of variables including the inherent state variables. M is a set of input messages, O is a set of output messages and Q is a message input queue. $F_S$ is a state transition function ($F_S$: S x M x V -> S), $F_O$ is an output function ($F_O$: S x M x V -> O) and $F_V$ is a set of actions performed during a specific state transition.

An AE can move to a new node. A *stable state* is a state of the *Main Function* where an AE's functionality can move safely and be re-instantiated in a new node based on the restoration of EFSM state, variables, and queued messages. *Strategist* is used by the *Main Function* to select appropriate actions. The *Main Function* will regularly

- Compare the condition part of the policies with inherent state variables, and will
- Activate the *Strategist* if a condition is met, which returns an action to be used by the *Main Function*

## B. *Strategist*

*Strategist* selects appropriate actions to be used by the *Main Function*. The *Strategist* behavior is based on a *Reasoning Machine (RM)* model, extended from [5, 6]. It can be triggered by one condition at a time. It will execute all policies related to a condition. The RM model R is defined as:

$$R \equiv \{ \Theta, \Phi, \Pi, \xi \} \qquad (2)$$

where Θ is a set of query expressions containing variables, Φ is a generic *reasoning procedure,* Π is a set of policies, and ξ is the strategist data including the inherent states from the *Main Function* and from other AEs, and available nodes and their capabilities.

$$\xi \equiv (\overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I, \hat{C}_{A,n};\ n=[1, N]) \qquad (3)$$
$$\Pi \equiv \{ p_i \} \qquad (4)$$
$$p_i \equiv (\Sigma_i, X_i, A_i) \qquad (5)$$
$$\Sigma_i \equiv \text{Expression}(\overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I) \qquad (6)$$
$$X_i \equiv \text{Expression}(\overline{S_R}, \hat{S}_R, \overline{C_R}, \hat{C}_R, I_R, \overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I, \hat{C}_{A,n};\ n=[1, N], \Gamma) \qquad (7)$$

A policy $p_i$ has conditions $\Sigma_i$, constraints $X_i$ and actions $A_i$. The condition is an expression of the inherent states from the *Main Function* and from other AEs. The constraint is an expression of required functionality and performance of services and capabilities, required service incomes, the inherent states from the *Main Function* and from other AEs, available nodes and their capabilities, as well as system time (Γ).

The reasoning procedure is applied to select appropriate actions with maximum accumulated rewards. It is based on *Equivalent transformation (ET)* [8], which solves a given problem by finding values for the variables of the queries. The conditions, constraints and actions can have variables. The result of the reasoning procedure can, in addition to actions, give instantiated variables.

## C. *Judge*

*Judge* gives rewards to actions to be selected by the *Strategist*. The reward is a numeric value based on the ability to move towards a state with goal performance and income measures. The rewards will be accumulated over a period of time. The *Judge* behavior is based on a *Learning Machine (LM)* model L defined as:

$$L \equiv \{ \Omega, \Lambda, \Psi, \zeta \} \qquad (8)$$

where Ω is a set of *goals*, Λ is a generic *rewarding procedure*, Ψ is a *reward database* storing the accumulated rewards of actions, and ζ is the judge data including the inherent states from the *Main Function* and from other AEs. We further have:

$$\zeta \equiv (\overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I) \qquad (9)$$
$$\Omega \equiv \{ g_k \} \qquad (10)$$
$$g_k \equiv (d_k, \omega_k) \qquad (11)$$

A goal $g_k$ has goal expression $d_k$ and weight $\omega_k$. The sum of the goal weights is equal to 1. At time t, the rewarding procedure will calculate the reward of an action $a_i$, which was applied at time t-1 as:

$$\text{reward}(a_i, i_{k,t-1}, d_k) = (\Delta(i_{k,t}, i_{k,t-1})/\Delta(d_k, i_{k,t-1})) * \omega_k - \text{cost}(a_i) \qquad (12)$$

where $i_{k,t-1}$ and $i_{k,t}$ are an inherent state measure before and after applying the action for an monitoring interval [t-1, t], $i_k \in \zeta$ and $d_k$ is an associated goal required measures. $\Delta(i_{k,t},i_{k,t-1})$ is the difference between $i_{k,t}$ and $i_{k,t-1}$. $\Delta(d_k,i_{k,t-1})$ is the difference between $d_k$ and $i_{k,t-1}$. $\omega_k$ is the goal weight and $\mathrm{cost}(a_i)$ is the operation cost of $a_i$.

The accumulated reward of an action $a_i$, accumulated-_reward($a_i$,$i_{k,t-1}$,$d_k$), is then the sum of the rewards of $a_i$, for an inherent state measure $i_{k,t-1}$ and a goal required measure $d_k$.

## D. *Communicator*

*Communicator* handles message sending and receiving on behalf of the *Main Function*. The *Communicator* behavior is based on the EFSM model in (1). Other AEs can subscribe to the inherent state variables of an AE. The *Communicator* will manage *subscription* messages and will send inherent state variables to other AEs on behalf of the *Main Function*.

The *Communicator* also handles the registration function on behalf of the *Main Function*. Registration message is sent to *Registry (REG)* (see Section V) that is an important AE that records the life-cycle state of AEs. The registration message contains IP address of the AE.

The *Communicator* will regularly broadcast *heartbeat message*, which is used to indicate that an AE is alive. The heartbeat messages are monitored by *Life Monitor AE (LMO)* (see Section V). In addition, the *Communicator* will inform REG about changes in the life-cycle state of the AE (see Section IV.E). REG will broadcast the changes to other AEs that subscribe to such updates.

## E. *Autonomic Element Life Cycle*

The combined states of an AE during its life-cycle are defined follows:

- *Initial state*: An AE is instantiated in a node where there are capabilities and capability functionalities and performances as required.
- *Registering state*: An AE registers to REG.
- *Normal-Active state*: An AE provides services with normal functionality and QoS level.
- *Degraded-Active state*: If in the Normal-Active-State the capabilities are less than required and results in degraded functionality and QoS, the life-cycle state will change to Degraded-Active state. In this state, some actions selected by the *Strategist* can be taken to upgrade the capabilities and performances. From both the Normal-Active state and the Degraded-Active state, the life-cycle state can change to Moving, Suspended or Terminated.
- *Moving state*: An AE's functionality is being moved and re-instantiated in a new node. A move can only take place if the EFSM states are stable (see Section IV.A).
- *Suspended state*: An AE is suspended, i.e., by an action selected by the *Strategist*, which means that it stops executing current behavior specifications. An AE will release its allocated capabilities. At this state, an AE can start executing new behavior specifications. This makes an AE goes back to the Normal-Active state.
- *Terminated state*: Other AEs detect that an AE's heartbeat message is lost or an AE could not be reached because of some unintentional reasons, e.g., the hardware failure. REG is informed about this by LMO or the other AEs, then REG records that an AE is terminated.

## V. Autonomic Element-Based Service Functionality Architecture

The service functionalities required for service configuration are constituted by *AEs* and repositories, as illustrated in Figure 3. The AE responsibilities and the repositories are described below.

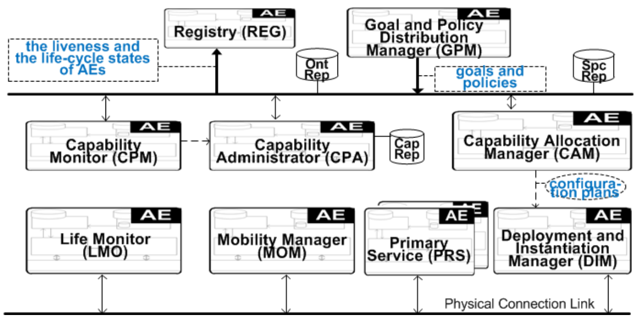
the liveness and the life-cycle states of AEs
Registry (REG)
AE
Goal and Policy Distribution Manager (GPM)
Ont Rep
goals and policies
Spc Rep
Capability Monitor (CPM)
Capability Administrator (CPA)
Cap Rep
Capability Allocation Manager (CAM)
configura-tion plans
Life Monitor (LMO)
Mobility Manager (MOM)
Primary Service (PRS)
Deployment and Instantiation Manager (DIM)
Physical Connection Link

Figure 3. Autonomic Element-Based Service Functionality Architecture. Solid arrows indicate the physical connections of AEs and dashed arrows represent the message flows between AEs.

- *Primary Service (PRS)* provides ordinary user services.
- *Registry (REG)*, as already mentioned in Section IV.D, is responsible for AE registration.
- *Goal and Policy Distribution Manager (GPM)* distributes goals and policies to corresponding AEs.
- *Life Monitor (LMO)* observes the liveness of AEs by listening to heartbeat messages from AEs. LMO regularly updates the liveness of AEs to REG.
- *Capability Administrator (CPA)* maintains and provides data about capabilities and their functionalities and performances in available nodes.
- *Capability Monitor (CPM)* monitors capabilities and sends updates to CPA.
- *Capability Allocation Manager (CAM)* generates (re-) configuration plans for AEs to be instantiated in nodes. CAM fetches the capability requirements and retrieves the capabilities from CPA. A configuration plan defines in which node an AE should execute. Configuration plans are generated based on capability requirements and policies. In addition, CAM allocates capabilities to AEs. The allocation depends on the capability structure

and optimization criteria which can be specified in the policies.

- *Deployment and Instantiation Manager* (*DIM*) executes the configuration plan. It creates AEs in the defined nodes and assigns the behavior specifications.
- *Mobility Manager (MOM)* supports an AE when it is moved and re-instantiated in a new node. The move can be related to failures or insufficient capability performances. MOM broadcasts messages to inform other AEs when an AE's functionality is suspended or is resumed. MOM also handles an AE's connections by getting input messages on behalf of an AE and forwarding them to such AE when it is already re-instantiated.
- *Ontology Repository (OntRep)* stores the goal and policy concepts as well as the related capability and service concepts.
- *Service Specification Repository (SpcRep)* stores the AE behavior specifications and the capability requirements.
- *Capability Repository (CapRep)* stores data about available nodes and their capabilities.

## VI. CASE STUDY

A music video streaming system is presented with the intention to demonstrate the *Strategist* and *Judge* solution in the proposed autonomic framework. The system is constituted by the AEs as defined in Section V. The Primary Service AEs are *Streaming Manager (STM)* and *Streaming Client (STC)*. An STM, executing on a *media streaming server (MS)*, streams the music video files to STCs. An STC is associated with an SLA class, which defines *required streaming throughput*, *price for the service* and *service provider penalties* if the agreed QoS cannot be met. Two SLA classes are applied: *premium (P)* and *ordinary (O)*. An STC is denoted by its SLA class as $STC_P$ or $STC_O$. Each SLA class has different required throughput (X); the $STC_P$ required throughput ($X_P$) can be 1Mbps or 600Kbps for high-resolution and degraded fair-resolution videos, while the $STC_O$ required throughput ($X_O$) is 500Kbps for low-resolution videos. Prices and penalties will be defined later.

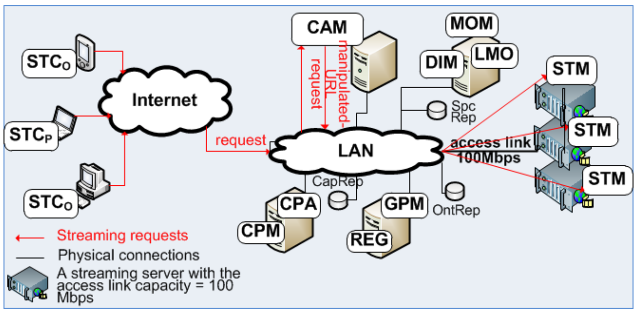

Figure 4. Streaming system example.

Figure 4 illustrates the streaming system. In this case study, CAM will accept the streaming requests on behalf of STMs. CAM will decide which an STM can serve the requests, or CAM may put them in waiting queues. CAM can also instantiate a new STM in an available MS that there is no executing STM.

The MS's required access link capacity ($C_{R,AL}$) is set to 100 Mbps. The number of STCs that can use the service at a time is limited by the MS access link capacity. When the required streaming throughput cannot be provided, a STC needs to wait until some connected requests have finished using the service. An $STC_O$ can be disconnected, while an $STC_P$ may have to degrade the video resolution. The service provider will pay penalties in case of waiting and disconnection of the STC. These penalty and price functions are given in Table II. A cost unit is the price paid by an ordinary client for one second streaming of the rate 500Kpbs. The price function for using the service is M(SLA_Class,X) (cost units/second). The penalty function for waiting is $P_{WAIT}$(SLA_Class) (cost units/second), and the penalty function for disconnection is $P_{DISC}$(SLA_Class) (cost units/connection).

Note that, the case study and all values, set in Table II, are same as our previous work [5, 6] in order to compare between the proposed and the previous model. The comparison results are in subsection VI.B.1.

TABLE II. THE PRICE AND PENALTY FUNCTIONS

| | **$STC_O$ ($X_O$=500Kbps)** | **$STC_P$ ($X_P$=600Kbps)** | **$STC_P$ ($X_P$=1Mbps)** |
|---|---|---|---|
| M(SLA_Class,X)/s | 1 | 1.875 | 2 |
| $P_{WAIT}$(SLA_Class)/s | 5 | 10 | 10 |
| $P_{DISC}$(SLA_Class)/ Connection | 10 | - | - |

The complete set of actions A in this case study is:

$$A = \{a_D, a_B, a_N, a_I, a_R, a_T, a_M\} \qquad (13)$$

A subset of A, Á, is defined as: $Á = A - \{a_M\}$. $a_D$ is to disconnect the ordinary clients, $a_B$ is to decrease the throughput of the premium clients, $a_N$ is to instantiate a MS, $a_I$ is to instantiate a new STM, $a_R$ is to disconnect a MS, $a_T$ is to terminate an STM and $a_M$ is to move connected client sessions from an STM to another STM. These actions are selected by the *Strategist* of CAM. CAM executes $a_N$, $a_I$, $a_R$ and $a_T$, while CAM suggests $a_D$, $a_B$ and $a_M$ to STMs.

In this case study, the considered capability is the MS access link. The required and inherent capability performance sets are denoted as $\hat{C}_R \equiv \{C_{R,AL}\}$ and $\hat{C}_I \equiv \{C_{I,AL}\}$, where $C_{R,AL}$ is the required access link capacity, and

$C_{I,AL}$ is the available access link capacity. The inherent service performance set $\hat{S}_I$ consists of the number of connected and waiting premium and ordinary clients ($N_{Con,P}$, $N_{Con,O}$, $N_{Wait,P}$, $N_{Wait,O}$), the number of disconnected ordinary clients ($N_{Disc,O}$), the number of MS ($N_{Node}$), the service time and waiting time of premium and ordinary clients ($T_{Serv,P}$, $T_{Serv,O}$, $T_{Wait,P}$, $T_{Wait,O}$). These values as well as the inherent service income ($I_I$) are observed per a monitoring interval Δ. The service income is defined as:

$$I_I = M(STC_O,X_O)*T_{Serv,O} + M(STC_P,X_P)*T_{Serv,P} - P_{WAIT}(STC_O)*T_{Wait,O} - P_{WAIT}(STC_P)*T_{Wait,P} - P_{DISC}(STC_O)*N_{Disc,O} - P_{Ser}*N_{Node}*\Delta \quad (14)$$

where $P_{Ser}$ is the cost function for adding a new MS which is 150 units/second per node, while M(SLA_Class,X), $P_{WAIT}$(SLA_Class) and $P_{DISC}$(SLA_Class) are as already defined in Table II.

## A. *RM and LM Specification*

In this case study, CAM plays an important role. Its RM specification is defined as follows:

$$R_{CAM} \equiv \{ \Theta_{CAM}, \Phi, \Pi_{CAM}, \xi_{CAM} \} \quad (15)$$

$\Pi_{CAM}$ consists of five policies ($p_1$-$p_5$) as presented in Appendix. A policy defines some actions in the set A in (13).

The LM specification of CAM is defined as follows:

$$L_{CAM} \equiv \{ \Omega_{CAM}, \Lambda, \Psi_{CAM}, \zeta_{CAM} \} \quad (16)$$
$$\Omega_{CAM} \equiv \{ g_1, g_2\} \quad (17)$$
$$g_1 \equiv (d_1: I_R > 0, \omega_1: 0.8) \quad (18)$$
$$g_2 \equiv (d_2: T_{Wait} < \Delta, \omega_2: 0.2) \quad (19)$$

where $I_R$ is the required service income, and $T_{Wait}$ is the sum of the waiting time of premium and ordinary clients. These goals are set in order to gain high income and to avoid high waiting time. The policy $p_1$-$p_5$ can be used when the required service income is not met, while the policy $p_1$-$p_3$ are used when the waiting time is higher than expected.

## B. *Experiments and Results*

Two set of experiments are presented. In B.1) Á is used for the comparison between the proposed and a previous model. In B.2) different action sets A and Á are used to study the proposed model. The accumulated service income and the accumulated waiting time results are illustrated in both experiment sets.

The request arrivals are modeled as a Poisson process with an arrival intensity parameter $\lambda_{SLA\_Class}$. The duration of streaming connections $d_{SLA\_Class}$ is constant and is set to 10 minutes. The traffic per MS access link $\rho$ is defined as:

$$\rho = ((\lambda_P*d_P*X_P) + (\lambda_O*d_O*X_O))/ (N_{Node}*C_{I,AL}) \quad (20)$$

The monitoring interval Δ is 1 minute. The STCs will stop waiting and there is no penalty for waiting after 10 minutes. The number of available MS = 3. Initially, only one STM is instantiated.

### *1) Comparison between the proposed and a previous model*

Our previous work [5, 6] presented an adaptation mechanism executed by a Reasoning Machine, which uses policies and goal to manage the adaptable systems. In the previous model, a policy consists of constraints and actions, and it is not associated with any specific conditions. So all defined policies will be executed when the systems are entering a reasoning condition. There is only one reasoning condition defined, i.e., the number of waiting clients > 0. In addition, only one goal based on the service income is used. The action is then rewarded by *the goodness score (QoX)* that is calculated by the percentage of the increased or decreased service income.

In this section, three different cases of ρ are illustrated: *a) ρ<1*, *b) ρ=1* and *c) ρ=1.15 (ρ>1)*. The $STC_P$ request arrivals intensity ($\lambda_P$) is set to 25%, 50% and 80% of the total arrival intensity.

For case *a) ρ<1*, both models have the same behaviors. If ρ>0.5, they used {$a_N$, $a_I$} to instantiate a MS and to instantiate a new STM, otherwise they just disconnected the ordinary clients or decreased the throughput of the premium clients. The accumulated service income and waiting time of both models are almost the same.

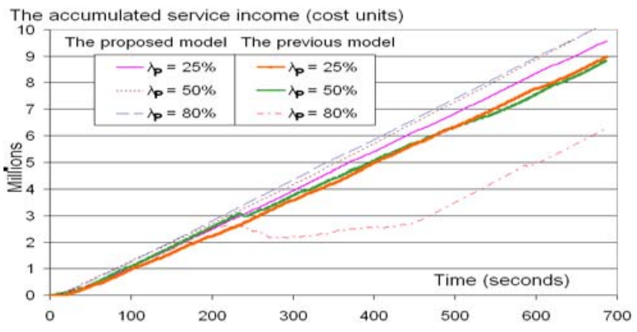

Figure 5. The accumulated service income when ρ = 1.

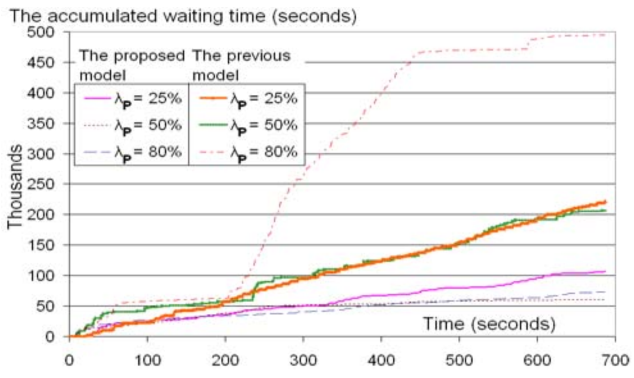

Figure 6. The accumulated waiting time when ρ = 1.

For case *b) ρ=1,* the proposed model produced higher accumulated service income and lower accumulated waiting time independent of $\lambda_P$, as depicted in Figure 5 and 6. This is because in the previous model $\{a_T, a_R\}$, which terminate an STM and disconnect a MS consecutively, was used when the number of waiting clients was little more than zero. So that, the number of MS was lower than proper required amount. It results in decreasing service income and increasing waiting time. In the proposed model, $\{a_T, a_R\}$ might be used only if the inherent service income <= 0; however, it does not happen when $\rho = 1$.

For case *c) ρ=1.15 (ρ>1)*, the proposed model also produced higher accumulated service income and lower accumulated waiting time. Figure 7 and 8 illustrates the accumulated service income and the accumulated waiting time for this case. The proposed model produced better results, because it took the actions to adapt the system more often than the previous model. When $\rho>1$, the service income could be less than 0 because of the waiting penalty. So that, in the proposed model the actions were applied both when the service income < 0 and when the waiting time > Δ, while the previous model the actions were applied only when the number of waiting clients > 0. However, when $\lambda_P$ = 80% the traffic was too overloaded, and the accumulated service income was less than zero in both models.

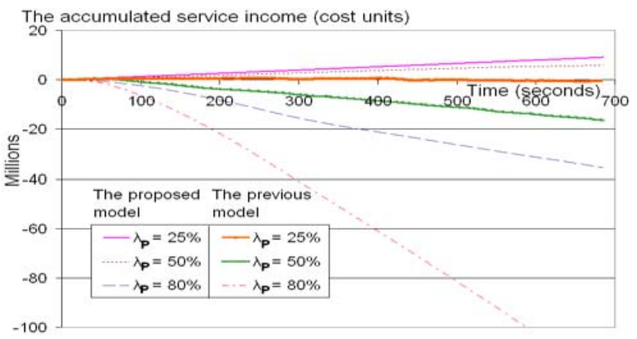

Figure 7. The accumulated service income when ρ = 1.15.

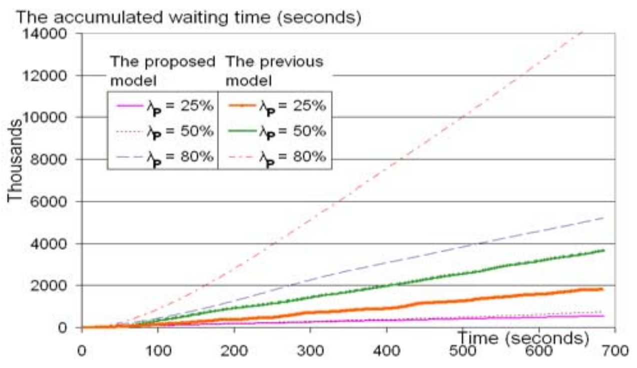

Figure 8. The accumulated waiting time when ρ = 1.15.

2) *Comparison between different action sets*

In this section, we compare three cases (I-III) of the new proposed model. In Case I the complete set of actions A is used, while in Case II the subset Á is used. For the last case, the complete set of actions A is also used, but there is no *Judge* component in the AEs so the actions are not rewarded. The traffics that were simulated for this scenario are relative to a function of time. The time with ρ at a fixed level, denoted as the *ρ period*, is set to 30 minutes. ρ varies from 0.2 to 1.2. $\lambda_P$ is set to 50% of the total arrival intensity.

Figure 9 and 10 shows the accumulated service income and the accumulated waiting time of three cases. The brown line in these figures shows the variation of ρ. In Case I, the system learned that $\{a_M, a_T$ and $a_R\}$, which move connected STC sessions, terminate an STM and disconnect a MS consecutively, is efficient to adapt the system when ρ drops and then the required service income is not met. As a result, Case I could produce the highest accumulated service income and the lowest accumulated waiting time. For the last case, the actions were selected randomly and they were not appropriate to the states of unwanted service income and the waiting time. So, the accumulated service income of Case III was the lowest, while the accumulated waiting time was the highest.

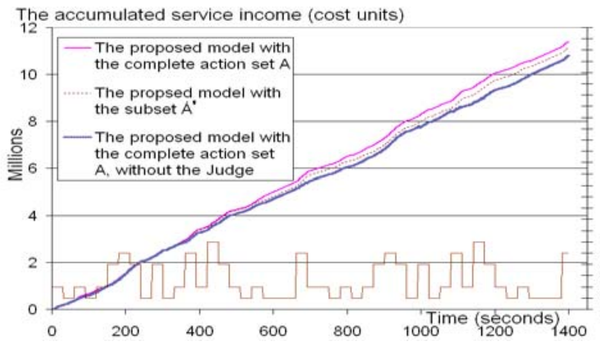

Figure 9. The accumulated service income for various ρ.

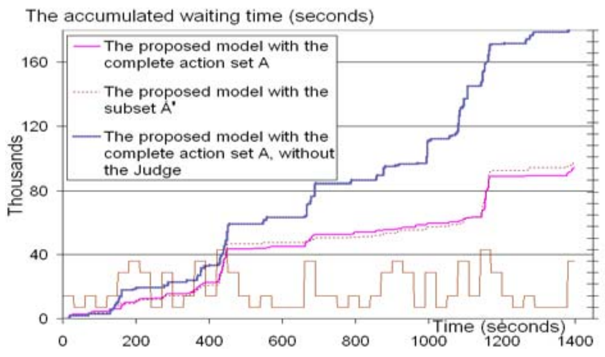

Figure 10. The accumulated waiting time for various ρ.

## VII. Related Work

Existing service system frameworks that support run-time self-management and adaptation can be classified based on the way in which the management and adaptation functionalities are specified. The functionalities can be *statically* or *dynamically* specified. Some works propose to use templates [9] or adaptation classes [10] to statically specify these functionalities. However, the static approach lacks flexibility. All the possible adaptation must be known a beginning, and if new adaptations are required, the systems must be re-complied. Our work expresses the service management functionality for the adaptable service systems in the form of the EFSM, RM and LM specification, to be dynamically modified, added and removed at run-time. When using the EFSM specification, an *update* of changes is done by deployment of the whole specification. When using the RM and LM specification, only incremental changes of the *policies* and *goals* are deployed. However, the complete policy and goal based functionality need to be validated off-line before the deployment of the incremental changes.

There are several works that use the policies to specify the adaptation, such as [11], [5, 6], [12-15]. Accord [11] is a framework that can formulate autonomic applications as dynamic composition of AEs, with the use of policies to describe the adaptation of functional behaviors of AEs and interactions between them. However, our approach and the rest go beyond the use of policy for the specification by adding mechanisms to adapt policies or the way of using policies. Such policy adaptation can be grouped into three categories: 1) changing the policy parameters, considered in [5, 6, 12, 13]; 2) enabling/disabling a policy, found in [5, 6, 12]; 3) using techniques to select the most suitable policy and action; for instance, rewarding policies and their actions, presented in [14, 15].

Our approach is an instance of the first as well as the third category as [14, 15]. Tesauro et al. [14] presented a hybrid reinforcement technique used for resource allocation in multi-application data centers. This technique is to select optimal policies that can maximize rewards. Mesnier et al. [15] used decision trees to select accurate policies in storage systems. These policy adaptation techniques have only been applied to a single element, while our approach is potentially used in multi-autonomic elements.

## VIII. Conclusions

This paper proposed an autonomic framework for adaptable service systems. The framework solution consists of *Goal-based Policy Ontology* and *Autonomic Element (AE) Model*. The Ontology defines common concepts of *goal*, *policy* and *inherent state*. AEs are generic component that can be used to realize any functionality. An AE is constituted by *Main Function*, *Strategist*, *Judge* and *Communicator* modules. The functionality of an AE is realized by two Extended Finite State Machines (EFSM), one Reasoning Machine (RM) and one Learning Machine (LM). EFSM behavior, as well as goals and policies can be modified flexibly during run-time. For attaining a specific functionality, specific EFSM, RM and LM functionality must be defined. In this paper, specific AEs handling service management functionality is proposed.

A case study is presented with focus on the Capability Allocation Manager (CAM). The experimental results show that the proposed model can produce higher service income and less waiting time than a previous model. In the proposed model, the actions are used appropriately under the associated goals and required goal measures. Moreover, it is possible to apply several goals, which each are weighed differently, depending on its importance. New actions can be added, and when there are more actions the system may reach the goals quicker.

## References

[1] P. Thongtra and F. A. Aagesen. Capability Ontology in Adaptable Service System Framework. In Proc. of 5th Int. Multi-Conference on Computing in the Global Information Technology, Spain, Sep 2010.

[2] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. IEEE Computer Society, January 2003, pp. 41-47.

[3] S. White, J. Hanson, I. Whalley, D. Chess, and J. Kephart. An architectural approach to autonomic computing. In Proc. of 1st IEEE Int. Conf. on autonomic computing, New York, May 2004, pp. 2–9.

[4] P. Thongtra and F. A. Aagesen. An Adaptable Capability Monitoring System. In Proc. of 6th Int. Conference on Networking and Services (ICNS 2010), Mexico, March, 2010.

[5] P. Supadulchai and F. A. Aagesen. Policy-based Adaptable Service Systems Architecture. In Proc. of 21st IEEE Int. Conf. on Advanced Information Networking and Applications (AINA'07), Canada, 2007.

[6] P. Supadulchai, F. A. Aagesen and P. Thongtra. Towards Policy-Supported Adaptable Service Systems. EUNICE 13th EUNICE Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services. Lecture Notes in Computer Science (LCNS) 4606, pp 128-140.

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

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

[9] F. Berman, R. Wolski, H. Casanova, et al. Adaptive computing on the grid using AppLeS. In IEEE Trans. Parallel Distrib. Syst., vol. 14, no. 4, pp. 369–382, Apr. 2003.

[10] P. Boinot, R. Marlet, J. Noy´e, G. Muller, and C. Cosell. A declarative approach for designing and developing adaptive components. In Proc. of the 15th IEEE Int. Conf. on Automated Software Engineering, 2000.

[11] H. Liu and M. Parashar. Accord: a programming framework for autonomic applications. In IEEE Trans. on System, Man, and Cybernetics, vol. 36, pp. 341–352, 2006.

[12] L. Lymberopoulos, E.C. Lupu and M.S. Sloman. An Adaptive Policy-Based Framework for Network Services Management. In Journal of Networks and Systems Management, vol. 11, pp. 277–303, 2003.

[13] K. Yoshihara, M. Isomura, and H. Horiuchi. Distributed Policy-based Management Enabling Policy Adaptation on Monitoring using Active Network Technology. In Proc. of 12th IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management, France, Oct 2001.

[14] G. Tesauro, R. Das, N.K. Jong, and M.N. Bennani. A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation. In Proc. of 3rd IEEE Int. Conf. on Autonomic Computing (ICAC'06), Ireland, Jun 2006, pp. 65–73.

[15] M. Mesnier, E. Thereska, D. Ellard, G.R. Ganger, G.R., and M. Seltzer. File classification in self-* storage systems. In Proc. of Int. Conf. on Autonomic Computing (ICAC-04), pp. 44–51.

[16] W3C, "OWL Web Ontology Language Overview," 2004 Available at: http://www.w3.org/TR/owl-features/

[17] V. Wuwonse and M. Yoshikawa. Towards a language for metadata schemas for interoperability. In Proc. of 4th Int. Conf. on Dublin Core and Metadata Applications, China, 2004.

## APPENDIX

The appendix includes the policy specifications and a list of mathematical expressions found in this paper.

### A: POLICY SPECIFICATIONS

The policies as well as goals are expressed in OWL (Web Ontology Language) [16] and OWL/XDD (XML Declarative Description) [17], where the variables can be integrated with ordinary OWL elements. The variables are prefixed with the $ sign. In this paper, the policy is written in the form:

**Conditions:** Expressions_for_conditions,
**Constraints:** Expression_for_constraints,
**Actions:** {Action_ID},
**Operation cost:** Expression_for_operation_cost

Five policies ($p_1$-$p_5$) used in the case study are listed in Table III. The conditions can be the inherent service income $\$I_I <= 0$ and the waiting time $\$T_{Wait} >= \Delta$, where $\$T_{Wait} = \$T_{Wait,P} + \$T_{Wait,O}$.

TABLE III. THE POLICY SET

| | |
|---|---|
| $p_1$ | **Conditions:** $\$I_I <= 0$ or $\$T_{Wait} >= \Delta$, **Constraints:** $P_{WAIT}(STC_O) < P_{WAIT}(STC_P)$, **Actions:** $\{a_D\}$, **Operation Cost:** $P_{DISC}(STC_O)$ This policy can be read as: $a_D$ should be used to disconnect a list of $STC_O$ when $P_{WAIT}(STC_O) < P_{WAIT}(STC_P)$, and the number of $STC_O$ being disconnected is calculated from $X_{P,1Mbps}$ * $\$N_{Wait,P}$ / $X_O$. $a_D$ costs $P_{DISC}(STC_O)$ units. |
| $p_2$ | **Conditions:** $\$I_I <= 0$ or $\$T_{Wait} >= \Delta$, **Constraints:** $P_{WAIT}(STC_O) > M(STC_P,X_{P,1Mbps})-M(STC_P,X_{P,600Kbps})$, **Actions:** $\{a_B\}$, **Operation Cost:** $M(STC_P,X_{P,1Mbps})$ - $M(STC_P,X_{P,600Kbps})$ This policy can be read as: $a_B$ should be used to decrease the throughput of a list of $STC_P$ when $P_{WAIT}(STC_O) >$ $M(STC_P,X_{P,1Mbps})$ - $M(STC_P,X_{P,600Kbps})$, and the number of $STC_P$ to decrease the throughput is calculated from $X_O$ * $\$N_{Wait,O}$ / ($X_{P,1Mbps}$ - $X_{P,600Kbps}$). $a_B$ costs $M(STC_P,X_{P,1Mbps})$ - $M(STC_P,X_{P,600Kbps})$. |
| $p_3$ | **Conditions:** $\$I_I <= 0$ or $\$T_{Wait} >= \Delta$, **Constraints:** ($X_{P, 1Mbps}$ * $\$N_{Wait,P}$ + $X_O$ * $\$N_{Wait,O}$) / $C_{R,AL} > 0.1$, **Actions:** $\{a_N, a_I\}$, **Operation Cost:** $P_{Ser} * \Delta$ This policy can be read as: $a_N$ and $a_I$ should be used to instantiate a MS and to instantiate a new STM consecutively, when ($X_{P, 1Mbps}$ * $\$N_{Wait,P}$ + $X_O$ * $\$N_{Wait,O}$) / $C_{R,AL} > 0.1$. These actions $\{a_N, a_I\}$ cost $P_{Ser} * \Delta$. |
| $p_4$ | **Conditions:** $\$I_I <= 0$, **Constraints:** ($X_{P, 1Mbps}$ * $\$N_{Wait,P}$ + $X_O$ * $\$N_{Wait,O}$) / $C_{R,AL} < 0.1$, **Actions:** $\{a_T, a_R\}$, **Operation Cost:** $P_{DISC}(STC_O) + P_{WAIT}(STC_P) – P_{Ser} * \Delta$ This policy can be read as: $a_T$ and $a_R$ should be used to terminate an STM and to disconnect a MS consecutively, when ($X_{P, 1Mbps}$ * $\$N_{Wait,P}$ + $X_O$ * $\$N_{Wait,O}$) / $C_{R,AL} < 0.1$. These actions $\{a_T, a_R\}$ cost $P_{DISC}(STC_O) + P_{WAIT}(STC_P) – P_{Ser} * \Delta$. |
| $p_5$ | **Conditions:** $\$I_I <= 0$, **Constraints:** ($X_{P, 1Mbps}$ * $\$N_{Wait,P}$ + $X_O$ * $\$N_{Wait,O}$) / $C_{R,AL} < 0.1$, **Actions:** $\{a_M, a_T, a_R\}$, **Operation Cost:** - $P_{Ser} * \Delta$ This policy can be read as: $a_M$, $a_T$ and $a_R$ should be used to move connected STC sessions, to terminate an STM and to disconnect a MS consecutively, when ($X_{P, 1Mbps}$ * $\$N_{Wait,P}$ + $X_O$ * $\$N_{Wait,O}$) / $C_{R,AL} < 0.1$. These actions $\{a_M, a_T, a_R\}$ make profit = $P_{Ser} * \Delta$. |

### B: MATHEMATICAL EXPRESSIONS

Table IV lists all mathematical expressions. This table also expresses the relations to others expressions (Rel. to exp.) as well as the references to table (Ref. to tab.), where the notification used in the expression are defined.

TABLE IV. MATHMATICAL EXPRESSIONS.

| No. | Mathematical expressions | Rel. to exp. | Ref. to tab. |
|---|---|---|---|
| 1 | $E \equiv \{ S_M, S_I, S_S, V, M, O, Q, F_S, F_O, F_V \}$ | - | - |
| 2 | $R \equiv \{ \Theta, \Phi, \Pi, \xi \}$ | - | - |
| 3 | $\xi \equiv (\overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I, \hat{C}_{A,n}; n=[1, N])$ | 2 | I |
| 4 | $\Pi \equiv \{ p_i \}$ | 2 | - |
| 5 | $p_i \equiv (\Sigma_i, X_i, A_i)$ | 4 | - |
| 6 | $\Sigma_i \equiv \text{Expression}(\overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I)$ | 5 | I |
| 7 | $X_i \equiv \text{Expression}(\overline{S_R}, \hat{S}_R, \overline{C_R}, \hat{C}_R, I_R, \overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I, \hat{C}_{A,n}; n=[1, N], \Gamma)$ | 5 | I |
| 8 | $L \equiv \{ \Omega, \Lambda, \Psi, \zeta \}$ | - | - |
| 9 | $\zeta \equiv (\overline{S_I}, \hat{S}_I, \overline{C_I}, \hat{C}_I, I_I)$ | 8 | I |
| 10 | $\Omega \equiv \{ g_k \}$ | 8 | - |
| 11 | $g_k \equiv (d_k, \omega_k)$ | 10 | - |
| 12 | $\text{reward}(a_i,i_{k,t-1},d_k) = (\Delta(i_{k,t},i_{k,t-1})/\Delta(d_k,i_{k,t-1}))*\omega_k - \text{cost}(a_i)$ | 8 | |
| 13 | $A = \{a_D, a_B, a_N, a_I, a_R, a_T, a_M\}$ | 5 | - |
| 14 | $I_I = M(STC_O,X_O)*T_{Serv,O} + M(STC_P,X_P)*T_{Serv,P} – P_{WAIT}(STC_O)*T_{Wait,O} – P_{WAIT}(STC_P)*T_{Wait,P} – P_{DISC}(STC_O)*N_{Disc,O} – P_{Ser}*N_{Node}*\Delta$ | 3, 6, 7, 9 | II |
| 15 | $R_{CAM} \equiv \{ \Theta_{CAM}, \Phi, \Pi_{CAM}, \xi_{CAM} \}$ | 2 | - |
| 16 | $L_{CAM} \equiv \{ \Omega_{CAM}, \Lambda, \Psi_{CAM}, \zeta_{CAM} \}$ | 8 | - |
| 17 | $\Omega_{CAM} \equiv \{ g_1, g_2\}$ | 10, 16 | - |
| 18 | $g_1 \equiv (d_1: I_R > 0, \omega_1: 0.8)$ | 11, 17 | - |
| 19 | $g_2 \equiv (d_2: T_{Wait} < \Delta, \omega_2: 0.2)$ | 11, 17 | - |
| 20 | $\rho = ((\lambda_P*d_P*X_P) + (\lambda_O*d_O*X_O))/ (N_{Node}*C_{I,AL})$ | - | - |