An Enhanced Goal-Oriented Decision-Making Model for Self-Adaptive Systems

by

Manbeen Kohli

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2011

Abstract

The realization of decision-making models in self-adaptive systems is still a challenge. Current state-of-the-art self-adaptive systems contain decision-making processes that leverage policy-based, goal-based or architecture-based adaptation models. Policy-based models are well suited to represent mandatory/rigid adaptation requirements. However, they cannot efficiently represent complex or flexible adaptation requirements. Moreover, conflict detection and resolution in these models is non-trivial and extremely error prone. Goal-based models utilize game theory techniques for conflict resolution; choosing a winner requirement from several conflicting requirements using state-of-the-art voting algorithms. This technique only serves systems containing flexible adaptation requirements. Architecture-based models rely predominantly on built-in analytical models, which are not generic. This implies that they cannot be incorporated into other existing or new systems. Furthermore, these models cannot cope with run-time changes that were not accounted for in their formulation, which limits their usage to very specific systems.

In this dissertation, we have engineered a generic, configurable and enhanced goal-oriented decision-making model that addresses the above-mentioned shortcomings in policy-based, goal-based and architecture-based decision-making models. The model provides the ability to represent both mandatory/rigid and flexible requirements. Additionally, the model provides the ability to represent related flexible adaptation requirements as clusters, and allows for the representation of multiple clusters. These clusters enable the execution of multiple corrective actions simultaneously, thus allowing a self-adaptive system to satisfy mandatory/rigid and flexible requirements concurrently. None of the above-mentioned decision-making models have the ability to do so. Additionally, the model has been designed to include feedback control loops as first class entities in the adaptation process. This enables assessing the impact of a previously executed decision, so that better decisions can be made in the future, thus allowing the model to cope with run-time changes. Furthermore, the model provides the ability to detect and resolve conflicts amongst dependant adaptation requirements.

The realization of the decision-making model is extremely generic, flexible and extensible. It allows different voting algorithms to be specified for choosing a winner requirement for clusters of flexible adaptation requirements. Moreover, the implementation also allows for the specification of a wide variety of reinforcement learning algorithms to assess the impact of a previously executed decision. The implementation has been developed as a plug-in for a generic Java-based adaptation framework. It was tested using two case studies namely a news web application and an IP telephony system. Based on the obtained results, it was concluded that the model does improve the overall customer satisfaction level compared to a non-adaptive system. Moreover, it was also concluded that incorporating feedback control loops as first class entities yields better results as compared to a decision-making model based solely on policies or goals.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Ladan Tahvildari for all her guidance and support over these years. I am also extremely grateful to Dr. Mazeiar Salehie from the STAR research group (Dr. Tahvildari's reasearch team) for his guidance and valuable feedback throughout my research. Their advice and mentorship has made this thesis possible.

Moreover, I would like to thank my dissertation committee members: Dr. Kostas Kontogiannis and Dr. Otman Basir, for having accepted to take the time out of their busy schedules to read my thesis and provide me with comments and inspiring remarks.

Additionally, I would like to thank all the present and past members of the STAR research group at the university, especially Sen Li for his help with the case studies used in this thesis.

Furthermore, I would like to thank my family for their unconditional support, and my friend Deepti for her encouragement. Last but not the least, I would like to thank my friends and co-workers at Sybase (an SAP Company) for their support and encouragement.

Dedication

This thesis is dedicated to my son Sajeev. Thanks kiddo for making my life complete!

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The simultaneous explosion of information, the integration of technology, and the continuous evolution from software-intensive systems require new and innovative approaches for building, running, and managing software systems. A consequence of this continuous evolution is that software systems must become more versatile, flexible, resilient, dependable, robust, energy-efficient, recoverable, customizable, configurable, and self-optimizing by adapting to changing operational contexts, environments or system characteristics[10]. In 2001, IBM introduced the concept of *Autonomic Computing* as a solution to address this issue.

According to[25], autonomic computing refers to a computing environment that is capable of managing itself, and can dynamically adapt to changes in accordance with business policies and objectives. An autonomic system refers to a system with a control loop, which monitors itself and its environment, analyzes the situation, and takes actions to change either the environment or its behavior. Therefore, the system exhibits two basic characteristics: *self-awareness*, which means that the system is aware of its state and behavior, and *context-awareness*, which means that the system is aware of its operational environment[23].

IBM defines four features of these systems often referred to as the self-* properties, which display different characteristics of self-management[25]:

- **Self-Configuring**: A self-configuring system can adapt to changing conditions and adjust itself automatically. It also supports dynamic addition/removal of components to/from the system.
- **Self-Optimizing**: A self-optimizing system is capable of monitoring and measuring performance related parameters in varying conditions and optimizing its behavior in order to meet performance objectives.

- **Self-Healing**: A self-healing system posses the ability to detect, diagnose, and repair problems. This property may also enable the system to be proactive in detecting future failures and thus prevent their occurrence. It also improves software reliability and availability.

- **Self-Protecting**: A self-protecting system has the ability to detect malicious attacks and to defend itself against them.

Autonomic computing is a broad research area, which includes systems from different domains like hardware, robotics, networks, grid computing, and software. In the context of this research, we focus only on *self-adaptive* software systems, also known as self-adaptive systems in this thesis.

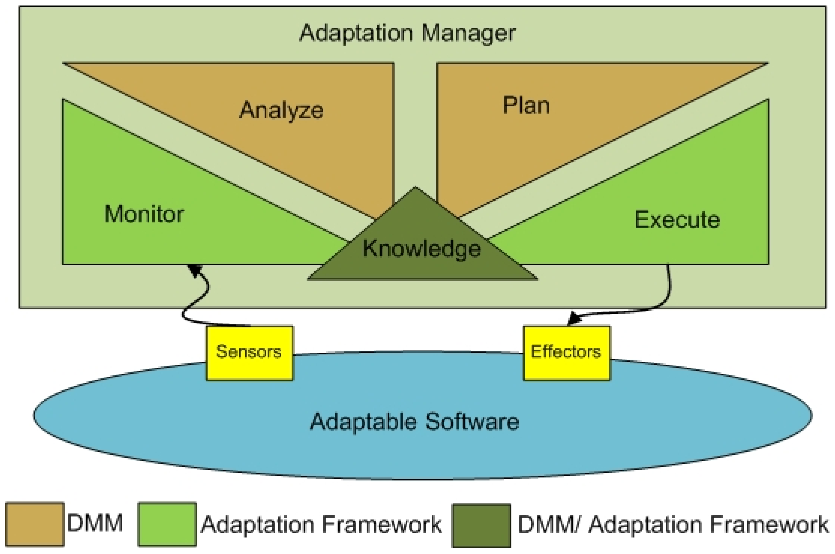


Figure 1.1: A Self-Adaptive Software System

Figure 1.1 shows the architecture of a typical self-adaptive software system. It consists of two major components namely the *adaptable software* and the *adaptation manager*.

- **Adaptable Software**: The application logic is implemented in adaptable software. The adaptable software exposes the required *sensors* and *effectors* for adaptation[27]. The sensors provide access to the current system attributes and the effectors can be utilized to modify the sytem attributes.

- **Adaptation Manager**: The adaptation manager realizes the monitoring, detecting, deciding and acting sub-processes to control the behavior of adaptable software.

Engineering of an adaptation manager involves designing and developing two critical components namely the *adaptation framework* and the *decision-making model.*

- **Adaptation Framework**: The adaptation framework provides the ability to periodically monitor the adaptable software, provides access to the data produced by the sensors, and facilitates the execution of an effector action that attempts to restore the system to normal.
- **Decision-Making Model (DMM)**: The decision making model interprets the data produced by the sensors to determine if the system is operating within its designated limits. If the system needs adaptation, the adaptation manager utilizes the decision making model to determine the action that must be performed by the effectors to bring the system back to normal.

The adaptation manager can either co-exist with the adaptable software or can be developed as an external process. A large number of existing solutions realize the adaptation process in an external adaptation engine[12]. Utilizing an external adaptation engine exemplifies the software design principle of *separation of concerns* wherein the the adaptable software processes the application logic and the adaptation manager controls the adaptation logic. Furthermore, it is the best technique available for adding self-* capabilities to existing legacy systems, due to its non-invasive nature.

1.1 Problem Description

Realization of the decision-making model in self-adaptive systems is still a challenge, as noted by Salehie et al[44] and McKinley et al[39]. The deciding process essentially needs to know about the adaptation requirements, business goals, articulated references provided by the system stakeholders. Additionally, the deciding process also needs complete information pertaining to the adaptable software to make decisions that maximize the business objectives and goals.

The adaptation requirements of a system can be divided into three categories namely *mandatory*, *negotiable* and *related negotiable* requirements. Mandatory requirements are those that must be satisfied by a system under all circumstances. On the other hand, negotiable requirements allow room for analyzing trade-offs during the decision-making process. Related negotiable requirements are the adaptation requirements that are similar; and can be analyzed as a *group/cluster.* A robust and comprehensive decision making model must provide an accurate and well-defined representation of all categories of adaptation requirements. Furthermore, the model must incorporate the business goals and stakeholder preferences.

Current state-of-the-art self-adaptive systems contain decision-making processes that leverage *policy-based, goal-based* or *application-specific architecture-based* adaptation techniques.

- **Policy-based Decision-Making Models**: Policy-based decision making models are well suited to represent systems with simple mandatory requirements. Representing negotiable or complex adaptation requirements with policies is extremely difficult and error prone. Moreover, detecting and resolving conflicts in these systems is an additional non-trivial challenge.

- **Goal-based Decision-Making Models**: Goal-based models utilize game theory techniques for conflict resolution; choosing a winner requirement amongst several conflicting requirements using state-of-the-art voting algorithms. Consequently these models are well suited to represent systems that contain only negotiable requirements. However, if all the requirements of a system are being dissatisfied at any given point in time, these models can only perform one corrective action at a time. This shortcoming limits the utilization of these systems in mission critical applications where several requirements may need to be satisfied concurrently.

- **Architecture-based Decision-Making Models**: Architecture-based models rely heavily on built-in application-specific analytical models. Since these models are application-specific, they cannot be incorporated into other existing legacy systems and/or new systems without a major re-engineering effort. Furthermore, these models cannot cope with the run-time changes that were not accounted for in their formulation. This implies that these models do not make sound decisions under uncertainity, which is an inherent nature of software systems.

The engineering of decision-making processes in self-adaptive software systems is a major challenge, especially if predictability and cost-effectiveness are desired. However, in other areas of engineering and nature there is a well-known, pervasive notion that could be potentially applied to software systems as well: the notion of feedback. Even though control engineering as well as feedback found in nature are not targeting software systems, mining the rich experiences of these fields and applying principles and findings to software-intensive adaptive systems is a most worthwhile and promising avenue of research for self-adaptive systems. To manage uncertainty in self-adaptive systems and their environments, we need to introduce feedback control loops as first-class entities in the decision-making model. The authors of [10] have observed that feedback control loops are often hidden, abstracted, or internalized when presenting the architecture of self-adaptive systems. However, the feedback behaviour of a self-adaptive system, which is realized with its control loops, is a crucial feature and hence should be elevated to a first-class entity in its modeling, design and implementation. Despite recent attention to self-adaptive systems (e.g. several

ICSE workshops), development and analysis methods for such systems do not yet provide sufficient explicit focus on the feedback control loops and their associated properties that almost inevitably control self-adaptation[10].

In this dissertation, we have engineered a generic, configurable and enhanced goal-oriented decision-making model that addresses the aforementioned shortcomings in policy-based, goal-based and architecture-based decision-making models. The model provides the ability to represent both mandatory/rigid and flexible requirements. Additionally, the model provides the ability to represent related flexible adaptation requirements as clusters, and allows for the representation of multiple clusters. These clusters enable the execution of multiple corrective actions simultaneously, thus allowing a self-adaptive system to satisfy multiple mandatory/rigid and flexible requirements concurrently. None of the aforementioned decision-making models have the ability to do so. Moreover, the model has been designed to include feedback control loops as first class entities in the adaptation process. This enables assessing the impact of a previously executed decision, so that better decisions can be made in the future, thus allowing the model to cope with run-time changes. Furthermore, the model provides the ability to detect and resolve conflicts amongst dependant adaptation requirements.

The realization of the decision-model is extremely generic, flexible and extensible. It allows different voting algorithms to be specified for choosing a winner requirement for clusters of flexible adaptation requirements. Moreover, the implementation also allows for the specification of a wide variety of reinforcement learning algorithms to assess the impact of a previously executed decision. The implementation has been developed as a plug-in for a generic Java-based adaptation framework.

1.2 Thesis Contribution

The major contributions of the enhanced goal oriented decision-making model developed in this thesis are as follows:

- **Comprehensive Representation of Adaptation Requirements**: The decision-making model has been engineered such that it can be used to represent different categories of adaptation requirements ranging from mandatory to negotiable requirements.

- **Concurrent Satisfaction of Multiple Unrelated Adaptation Requirements**: The decision-making model has been designed to enable representation of related flexible adaptation requirements as clusters. It also provides support for the representing multiple clusters. Consequently the model provides ability to concurrently execute multiple corrective actions and thus satisfy mutiple unrelated adaptation requirements simultaneously.

- **Incorporation of Feedback Control Loops as First Class Entities**: The decision-making model enables the incorporation of feedback control loops as first class entities in the decision-making process of a self-adaptive system. This enables assessing the impact of a previously executed decision, so that better decisions can be made in the future, thus allowing the model to cope with run-time changes.
- **Conflict Detection and Resolution**: The decision-making model provides a mechanism to detect and resolve conflicts between dependent adaptation requirements.
- **Ultimate Flexibility in Specification of Voting Algorithm**: The decision-making model has been engineered to be extremely flexible. It provides the ability to specify any voting algorithm to choose a winner amongst competing flexible requirements. The realization of the model contains some built-in voting algorithms. However, a user is free to develop any voting algorithm that implements a specified interface.
- **Ultimate Flexibility in Specification of Reinforcement Learning Algorithm**: The decision-making model has been engineered to be extremely flexible. It provides the ability to specify any reinforcement learning algorithm to asses the impact of a previously executed decision, so that better decisions can be made in the future. The realization of the model contains some built-in reinforcement learning algorithms. However, a user is free to develop any reinforcement learning algorithm that implements a specified interface.

The next section describes the organization of this dissertation.

1.3 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 presents a literature review of research related to this work. It outlines the different methodologies employed by decision-making models in self-adaptive systems and discusses the merits and drawbacks of each approach. In doing so we also highlight some of the research gaps in this area.

Chapter 3 provides an overview of the proposed enhanced decision-making model and its conceptual architecture. It first describes the components of a generic decision-making model, followed by the functional requirements of a well-designed and comprehensive decision-making model. Subsequently it describes the architecture of the decision-making model developed in this thesis. Lastly, it describes how a decision-making process can utilize the proposed decision-making model to ensure that the adaptation requirements are being satisfied by the system.

Chapter 4 describes the procedure for building an adaptation manager for self-adaptive systems consisting of the decision-making model developed in this thesis and StarMX - an adaptation framework developed for Java-based self-adaptive systems. It then describes the configuration information for the adaptation manager, the run-time behavior of the adaptation manager and the deployment options that can be used by the adaptation manager. Finally, we describe a six-step process for developing a self-managing system using the proposed decision-making model and a Java-based self-adaptive application framework.

Chapter 5 reports the conducted experimental studies that evaluate the effectiveness of our model. In the first and second sections, we describe our case studies, which are a news web application and a voice-over-IP system. We define the research objectives of the experiments and how they are analyzed. The last section provides a discussion on the results of the experiments and reports framework performance analysis results.

Finally, Chapter 6 finishes the thesis by drawing conclusions from the presented research. It discusses future directions for the research, and outlines some concluding remarks.

Chapter 2

Related Work

Over the past decade, researchers and practitioners have developed a variety of methodologies, frameworks, and technologies intended to support the construction of self-adaptive systems. Engineering the adaptation logic poses the greatest challenge in these systems and a significant amount of research is being conducted in this area. This chapter describes some of the pivotal research work done in this area.

We begin this chapter by categorizing the realization techniques for decision-making models used by self-adaptive systems. Next, we present an overview of each category and outline the pros and cons of each approach. Subsequently, we explore a new and upcoming research area namely the incorporation of feedback control loops as first-class entities in decision-making models of self-adaptive systems and present the challenges associated with doing so. We conclude this section by identifying the existing research gaps in the area of decision-making models.

2.1 Categorization of Decision-Making Models

Realization of the decision-making model in self-adaptive systems is still a challenge, as noted by Salehie et al[44] and McKinley et al[39]. The deciding process needs complete information pertaining to the adaptable software to make decisions that maximize the business objectives and goals. Additionally, the deciding process also needs to know about the articulated references provided by the system stakeholders and the adaptation requirements of the system. A robust and comprehensive decision making model must provide an accurate and well-defined representation of the business goals, stakeholder preferences and the adaptation requirements. In order to design and engineer an effective decision-making model, we execute the following steps:

- **Survey and Categorize**: We first survey existing projects in the area of self-adaptive software. We do this to enable categorization of the adaptation requirements of the system and their realization i.e. the decision-making model.

- **Analyze and Compare**: Next, we analyze the pros and cons of each category of decision-making models.

- **Identify Research Gaps**: Subsequently, we identify the research gaps in this area and design a decision-making model that addresses these gaps.

The projects in this section are selected from different academic and industrial sectors to capture main research trends in the broad area of self-adaptive software. The information is collected from many academic and industrial research projects. However, a few of them are selected to represent the major research ideas in this field. Space limitations, the diversity of ideas, and their impact on the field, are the concerns taken into account for the selection. Table 2.1 lists these projects.

Self-adaptive systems view the adaptation requirements of a system as objectives or goals that must be satisfied by the system. The adaptation requirements of a system essentially fall into three categories namely:

- **Mandatory Requirements**: Mandatory requirements are the rigid requirements that the adaptable system must absolutely satisfy under any circumstances. In a sense, they are similar to the *hardgoals* described by van Lamsweerde in [30]."If the load on a sytem goes above critical limits, then the system must gracefully shutdown to prvent any data loss that may occur due to a crash" is an example of a mandatory requirement.

- **Negotiable Requirements**: Negotiable requirements are similar to *softgoals*[30] and are flexible in a sense since they allow the evaluation of trade-offs during decision making. According to the Non-Functional Requirements (NFR) approach of software requirement engineering mentioned in [40], goal satisficing derives from the notion that goals are never totally achieved or not achieved. Thus, "softgoals are satisficed when there is sufficient positive and little negative evidence for this claim, and that they are unsatisficeable when there is sufficient negative evidence and little positive support for their satisficeability." A system may not be able to satisfy all the negotiable requirements at any given point in time and may have to compare the constituent requirements (using a voting algorithm) to choose a "winner" requirement. To facilitate choosing a "winner" requirement, priorities may need to be assigned to the negotiable requirements. An example of a system with negotiable requirements is as follows: Consider a news website with two requirements namely providing users with a quick response time and providing users with the best visuable experience i.e.

Table 2.1: Selected Self-Adaptive Software Projects

Project Name	**Description**
Rainbow[11]	Proposing an architecture-based adaptation framework consisting of an adaptation infrastructure and system-specific adaptation knowledge specified through policies.
KX[12]	A generic framework for collecting and interpreting application-specific behavioral data at run-time through sensors (probes) and gauges.
Accord[14]	Providing a programming framework for defining application context, autonomic elements, rules for the dynamic composition of elements, and an agent infrastructureto support rule enforcement.
TRAP[20]	A tool for using aspects and reflective technique for dynamic adaptation in Java,TRAP/J, and .Net framework, TRAP/.Net.
CASA[9]	Contract-based Adaptive Software Architecture (CASA) supports both application-level and and low-level (e.g. middleware) adaption actions through an external adaptation engine.
GAAM[44]	A goal-oriented decision-making model for self-adaptive systems using a weighted voting mechanism for the action selection process.
J3[15]	Providing a model-driven framework for application-level adaptation based on three modules J2EEML, JAdapt, and JFense respectively for modeling, interpreting and run-time management of self-adaptive J2EE applications.
[19]	A framework for adaptive policy-driven autonomic management using reinforcement learning methodologies.
FUSION[6]	A decision-making framework for self-adaptive systems that combines feature-orientation, supervised learning, and dynamic optimization.

the highest quality of text, videos and images. If a large number of users connect to the website concurrently, then satisfying both requirements may not be possible. This is because in order to provide a quick response time under heavy load, the server can display images of a lower quality, or may be even display a text only version of a web page to a user, but this would conflict with the requirement of providing the best visual experience. In this situation, the adaptation manager will need a choose a "winner" requiement to ensure healthy system operation.

- **Related Negotiable Requirements**: Related negotiable requirements are the adaptation requirements that are similar; and can be analyzed as a group or cluster. All of the requirements belonging to a particular group are related; and the requirements belonging to different groups are othrogonal. For example, consider a self-adaptive database systems that contain adaptation requirements related to its CPU utilization, and adapdation requirements related to its disk usage. Additionally, consider that these requirements are flexible, and that multiple requirements are not being satisfied at some given point in time. Since these requirements are orthogonal, the adaptation manager can easily choose one "winner" requirement from both categories, and thus satisfy multiple adaptation requirements concurrently.

Projects like Rainbow[11] or Accord[14] view the adaptation requirements as mandatory objectives which need to be satisfied under all circumstances. Other projects like GAAM[44] view the adaptation requirements as flexible goals or softgoals where trade-off analysis is incorporated in the decision-making process. This implies that the adaptation requirements are never completely satisfied or dissatisfied; the adaptation requirements are *satisficed*. The decision-making models employed by projects like Rainbow[11], CASA[9] and J3[15] use architecture-based adaptation and rely on analytical models for making adaptation decision. Based on this discussion, the decision-making models of a self-adaptive system can be categorized as follows:

- **Policy-based Decision-Making Models**: Policy-based decision making models levarage rules or policies for realization of the adaptation requirements.
- **Goal-based Decision-Making Models**: Goal-based models represent the adaptation requirements as goals. These models utilize game theory techniques for conflict resolution; choosing a winner requirement amongst several conflicting requirements using state-of-the-art voting algorithms.
- **Architecture-based Decision-Making Models**: Architecture-based decision-making models rely on analytical models for making adaptation decisions.

It should be noted that the above mentioned categories are not orthogonal. The decision-making model in the Rainbow framework[11] for instance leverages both policy-based and architecture-based adaptation techniques. The next sections describe the above mentioned categories in further details.

2.2 Policy-based Decision-Making Models

Policies can be considered as directives to an adaptation manager to use in order to meet adaptation requirements. Utilizing policy-based management as a means of describing the adaptation logic has received significant interest in recent years. Within self-adaptive systems, policies can be considered to fall into two main categories[24]:

- **Action Policies**: Action policies take the form of $IF(Condition) THEN(Action)$,where the policy specifies possible actions that could be taken whenever certain conditions occur; that is, the policy is violated. Action policies can be correlated to the decision-making model employed by action-based rational agents.
- **Utility Function Policies**: Utility function policies define an objective function that aims to model the behavior of the system at each possible state. They are more flexible than action policies and can be correlated to the decision-making model utilized by utility-based rational agents.

The next sections provide additional details about the aforementioned policy categories.

2.2.1 Action Policies

A significant number of researchers have used policy-based management employing action policies in the domain of self-adaptive systems. Action policies are mostly implemented using rule-based mechanisms and are used in a wide variety of commerical and academic projects. Microsoft SQL Server and IBM's DB2 Universal Database systems use action policies for self-tuning purposes.

The Accord framework is an academic project[14] which utilizes application context, definition of autonomic elements, dynamic composition rules/mechanisms, and an infrastructure for rule enforcement. The authors of[11] have developed a framework called Rainbow for architecture based adaptation which proposes an adaptation language to capture managers choices and their utility information. However, both these projects use deterministic rule-based mechanisms for deciding, and hence *cannot* address deliberative decision making and deciding under uncertainty.

Several languages have been developed by academia, industry, and standards bodies for implementing action policies for the security and access control, system management, and network administration domains. A few examples of action policy based languages are XACML (eXtensible Access Control Markup Language) from OASIS, Ponder from Imperial College, PDL (Policy Description Language) from Bell Labs, ACPL (Autonomic Computing Policy Language) from IBM[4].

The main advantage of using a policy-based decision-making model based exclusively on action policies is its simplicity. Action policies can be used as a simple technique to create a closed control loop such that appropriate actions are executed when special conditions are met. Policy-based configuration management using action policies has been used to reduce configuration errors leading to improved availability, performance, and system security in systems utilizing a few fundamental (but simple) non-conflicting policies.

However, there are certain drawbacks associated with using a policy-based decision-making model based exclusively on action policies. Generally, such a system will have at least as many policies defined as the number of attributes that are being monitored for the adaptation process. This implies that the action policy space must be at least as large as the attribute space. This increases the memory requirements of such systems by a significant amount, and thus may prevent action policies from being used as the exclusive decision making approach in resource constrained self-adaptive systems.

Furthermore, action policies are implemented to constrain the external environment, and the intent is for the policies to be individually implemented regardless of the other policies. Although in practice action policies may interact with each other and may conflict with each other, this is not the main intent of a policy-based system based on action policies[4]. In an environment where multiple sets of action policies may exist, and where at run-time multiple policies may be violated, policy selection is often based on statically configured policy priorities which an administrative user may have to explicitly specify. This exacerbates the complexities associated with the management, maintenance and evolution of such systems.

2.2.2 Utility Function Policies

The effective use of policies in the management of self-adaptive systems requires that the policies be captured and translated into actions within the system. Most previous research on the use of policies for implementation of the adaptation process has mainly focused on the specification and use "as is" within systems and where changes to policies are only possible through manual intervention[19].

The complex and stochastic nature of today's computing environments often make it impractical to obtain information that is both accurate and representative of all possible situations a system may encounter while interacting with its environment. As systems become more complex, relying on humans to encode rational behavior onto policies is definitely not the best way forward. It is imperative, therefore, that self-adaptive systems have mechanisms for adapting the use of policies in order to deal with not only the inherent human error, but also the changes in the configuration of the managed environment and the complexities due to unpredictability in workload characteristics.

In the context of where utility function policies are used to drive self-management, this requires having a system monitor its own use of policies to learn which policy actions are most effective in encountered situations. This information is then used to enable the system to learn from past experience, predict future actions and make appropriate trade-offs when selecting policy actions.

The authors of [19] have developed a framework for adaptive policy-driven autonomic management using reinforcement learning methodologies. To capture systems dynamics in the use of policies, a state-transition graph is used. The vertices of the graph are the states that a system can be in at any given point in time which is deduced based on the monitored attribtues of the system. The transitions of the graph are the actions performed by the system that transform the system from one state to another. Each action has a utility function value associated with it. The decision-making model uses these values for action selection, and updates the utility values of a previously executed action based on the current system state. The value is incremented for favorable states and decremented for unfavorable states.

One of the major advantages of decision-making models based on utility function policies is that the model of the environment is "learned" on-line and used, at each timestep, to improve the policy guiding the agents interaction with the environment. Moreover, the strategy for adapting the use of policies make use of a learning signal that is based only on the structure of the policies. This implies that the methology is generic and can be used in other existing/new software systems with minimal to no re-engineering effort.

A drawback of this approach is that model-based reinforcement learning mechanisms are often computationally demanding. Researchers are currently investigating the use of policies to guide the learning process and its use of computational resources. This ranges from the selection of the type of algorithm to be used, to the adjustment of algorithms parameters to meet the resource constraints imposed by the environment.

2.3 Goal-based Decision-Making Models

Goal-oriented requirements engineering is one of the common ways to model system requirements and to analyze the impact of variant design decisions on them. Goals and agents have been proposed as the best suited intentional concepts to capture the properties of complex, dynamic and adaptive systems in software requirements elicitation and analysis. It should be noted that the self-* properties used to describe self-adaptive systems align well with the non-functional requirements of a system. Therefore, as in goal-driven requirements models (e.g. SIG[31]), these properties can be mapped to quality goals, which in turn can be decomposed into smaller sub-goals. Agent-oriented software engineering (AOSE) methodologies[22], such as MaSE, Prometheus and Tropos, use goal models to capture stakeholders objectives the system under consideration should achieve[30].

Several researchers subscribe to the view that it is essential to explicitly represent system goals, and to incorporate these goals in the adaptation processes. The belief is that goal-based adaptation could not only be effective, but also be traceable and trustable. In [5], the authors identify a goal-oriented approach for engineering adaptivity requirements including monitoring, decision, and adaptation functionalities. In this four levels framework, each level corresponds to the objectives of a different stakeholder.

- **Level 1**: Level 1 comprises traditional requirements engineering, fulfilled by the system developer in order to elicit customers and users objectives.
- **Level 2**: Level 2 considers requirements engineering, fulfilled by the system itself at run-time in order to determine if and how to adapt.
- **Level 3**: Level 3 includes requirements engineering to determinate the system adaptation architecture.
- **Level 4**: Level 4 spans research on adaptation.

A significant amount of research on goal-based decision-making models has been focussed on the design and architecture i.e. on level 1 and level 3. On level 1,[21] proposes goal-oriented modelling of every possible system configuration in a distinct goal model (using KAOS). The KAOS methodology has been used for several industrial systems. Specifically, van Lamsweerde et al.[2] used the KAOS methodology to elaborate the requirements of a meeting scheduler and introduced refinement patterns in the context of KAOS[40]. These patterns were intended to capture commonly occurring situations when modeling software.

The work presented by the authors of [34] covers level 1 of this categorization and aims at defining the foundation necessary to address the implementation of level 2. Also [1] can be categorized at level 1. It enriches i* models to obtain a design-level view, aiming at a specification of autonomic systems. Annotations such as sequence, priority, and conditions are introduced for decompositions, while, expressions can define variable contribution to softgoals. The obtained models are a first step towards the goal model approach to get a more detailed system design.

Based on the literature review, some research efforts utilize a goal-based model in the decision-making process at run-time. For example, Kramer et al. discuss an architecture based adaptation approach, which utilizes a goal management layer[28]. This layer generates a reactive plan to satisfy the goals. Salehie et al. also work on an adaptation approach based on quality goals, in order to trace and satisfy these goals at run-time[43]. In another work Salehie et al. employed a weighted voting scheme for goal-based decision-making[44].

Goal-based decision-making models have several advantages over a decision-making model based exclusively on policies. Goals give a self-adaptive system the flexibility to choose the best actions under the current conditions. Goals are also easy to understand and relate to the overall system purpose. Moreover, the goal space in most cases is smaller than the attribute space, since multiple attributes can be associated with a single goal. This implies that each attribute definition does not necessarily require a separate goal definition since goals and attributes can share a many-to-one relationship. Furthermore, the goal-based management approach also simplifies the ability to detect and resolve conflicts.

A shortcoming exhibited by goal-based decision-making systems is that goals are relatively more complex to define, setup and process as compared to policies. Some goals are dependent upon system models and planning algorithms. When not all goals can be realized, the goals alone provide no guidance. In particular, it is unclear what the best way is to handle failure during adaptation execution, or whether it needs to be dealt with at all if one assumed a continuous adaptation cycle of monitor and control. Clearly, a proper treatment of failure must ensure that the adaptation process can recognize what failure state it is in and recover from that failure.

Another drawback of self-adaptive systems using goal-based decision-making models is the underlying assumption that all the adaptation requirements are flexible and can be represented as *softgoals*[40]. The adaptation requirements of a system may contain *both* mandatory and flexible requirements. Moreover, the flexible requirements may be *related*, i.e. a system may contain a *cluster* of related adaptation requirements, where the requirements belonging to a cluster may be treated as softgoals, and allow trade-off analysis for their *satisfication*[40]. Furthermore, a system may contain several such clusters of adaptation requirements. Currently neither a goal-based decision-making model nor a policy-based decision-making model can be used to realize the adaptation requirements of such a system.

2.4 Architecture-based Decision-Making Models

Architecture-based self-adaptive software focuses on using software architecture models as the central abstraction for decision-making and change enactment. The focus on explicit architectural models addresses the challenges and complications imposed by the multiple artifacts and levels of abstraction through which autonomous behavior can be expressed. A variety of research efforts have adopted this architecture-centric approach including contributions based on dynamic software architecture description languages (ADLs) as well as work based on dynamic distributed systems managed using explicit architectural models.

Oreizy et al. have pioneered the architecture-based approach to run-time adaptation and evolution management in their seminal work[38]. Garlan et al. have developed the Rainbow framework[11], a stylebased approach for developing reusable self-adaptive systems. Rainbow monitors a running system for violation of the invariant imposed by the architectural model, and applies the appropriate adaptation strategy to resolve such violations. The authors of [9] have developed an adaptation framework called Contract-based Adaptive Software Architecture (CASA) which supports both application-level and and low-level (e.g. middleware) adaption actions through an external adaptation engine. The aforementioned approaches and many others share three traits:

- Use analytical models for making adaptation decisions.
- Rely on architectural models for the analysis.
- Effect a new solution through architecture-based adaptation.

Existing self-adaptive frameworks require a software engineer to construct and utilize complex analytical models. This is because it is difficult foresee all of the changes in the environment, requirements, and systems operational profile at design-time[6]. Unfortunately, the majority of widely used analytical models have to painstakingly be customized to the unique characteristics of an application domain. Moreover, for any application-specific objective, an appropriate analytical model would have to be developed from scratch; a task that is often very difficult, when one considers the complexity of todays software systems, and the fact that most software engineers are not savvy mathematicians[6].

Additionally, using application based analytical models for self-adaptation is not without its set of challenges. These models when given the monitoring data obtained at run-time assess the systems ability to satisfy its objectives. The results produced by these models thus serve as indicators for making the adaptation decisions. Analytical models make simplifying assumptions or presume certain properties of the running system that may not bear out in practice. Furthermore, these models are specified at design-time and cannot cope with the run-time changes that were not accounted for in their formulation. These assumptions could make the analysis and hence the adaptation decisions inaccurate.

Table 2.2 summarizes the pros and cons of the different categories of decision-making models discussed in the previous sections. The next section describes a relatively new and upcoming research area with regards to decision-making models in self-adaptive systems.

Table 2.2: Comparison of Decision-Making Model (DMM) Categories

Category	**Examples**	**Advantages**	**Disadvantages**
Action Policy-based DMM	Rainbow[11] Accord[14]	• Easy to understand and implement. • Used in a large number of systems with simple mandatory adaptation requirements.	• Require large amount of memory since policy space greater than attribute space. • Cannot be used to represent complex/flexible requirements. • Unsuitable for conflict detection/resolution.
Utility Function Policy-based DMM	FUSION[6] [19]	• Model of environment learned "on-the-fly". Can improve on previously made decisions. • Policies independent of adaptable software.	• Model-based reinforcement learning mechanisms can potentially be computationally expensive.
Goal-based DMM	GAAM[44]	• Require less memory than action policies since goal space smaller than attribute space. • Better suited to conflict detection/resolution compared to policies.	• Cannot be used to represent mandatory requirements. • Cannot be used to satisfy multiple requirements concurrently.
Architecture-based DMM	Rainbow[11] CASA[9] TRAP[20] J3[15]	• Generic frameworks can be applied to legacy systems.	• Analytical models specified at design time cannot cope with unaccounted run-time changes.

2.5 Control Loops and Self-Adaptive Systems

Feedback control loops have been recognized as important factors in software process management and improvement or software evolution. Lehmans work on software evolution showed that "the software process constitutes a multilevel, multiloop feedback system and must be treated as such if major progress in its planning, control, and improvement is to be achieved." Therefore, any attempt to make parts of this "multiloop feedback system" self-adaptive necessarily also has to consider feedback control loops. To manage uncertainty in computing systems and their environments, we need to introduce feedback control loops to control the uncertainty. Feedback control loops provide the generic mechanism for self-adaptation. Positive feedback occurs when an initial change in a system is reinforced, which leads toward an amplification of the change. In contrast, negative feedback triggers a response that counteracts a perturbation.

The authors of [48] argue that to reason about uncertainty effectively, feedback control loops need to be made visible and first class entities. If the feedback control loops are invisible, we will not be able to identify which feedback control loops may have major impact on the overall system behavior and apply techniques to predict their possible severe effects. More seriously, we will neglect the proof obligations associated with the feedback, such as validating that the estimate of data derived from the sensors is sufficiently good, that the control strategy is appropriate to the problem, that all necessary corrections can be achieved with the available effector, that corrections will preserve global properties such as stability, and that time constraints will be satisfied[48]. Therefore, if feedback control loops are not visible we will not only fail to understand these systems but also fail to build them in such a manner that crucial properties for the adaptation behavior can be guaranteed.

The authors of [48] also mention the challenges associated with making feedback control loops first class entities in self-adaptive systems. Some of these challenges have been described below:

- **Modeling**: There should be modeling support to make the feedback control loops explicit and to expose self-adaptive properties so that the designer can reason about the system. The models have to capture what can be observed and what can be influenced.
- **Verification and Validation**: Development of self-adaptive systems requires techniques to validate the effects of feedback control loops.
- **Reengineering**: Today, most engineering issues for self-adaptive systems are approached from the perspective of greenfield development. However, many legacy applications can benefit from self-adaptive features. Reengineering of existing systems

with the goal of making them more self-adaptive in a cost-effective and principled manner poses an important challenge.

One of the realization methods for incorporating feedback control loops as visible first-class entities in the decision-making models of self-adaptive systems is through *machine learning*, specifically *reinforcement learning* techniques.

2.6 Machine Learning

Machine learning, a branch of artificial intelligence, is a scientific discipline concerned with the design and development of algorithms that allow systems to evolve behaviors based on empirical data. A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data. However, this is non-trivial and the difficulty lies in the fact that the set of all possible behaviors given all possible inputs is too large to be covered by the set of observed examples. Consequently, a learner must generalize from the given examples, so as to be able to produce a useful output in new cases.

Applications for machine learning include machine perception, computer vision, natural language processing, syntactic pattern recognition, search engines, medical diagnosis, bioinformatics, brain-machine interfaces and cheminformatics, detecting credit card fraud, stock market analysis, classifying DNA sequences, speech and handwriting recognition, object recognition in computer vision, game playing, software engineering, adaptive websites, robot locomotion, and structural health monitoring.

Machine learning algorithms are organized into a taxonomy, based on the desired outcome of the algorithm.

- **Supervised learning** generates a function that maps inputs to desired outputs. For example, in a classification problem, the learner approximates a function mapping a vector into classes by looking at input-output examples of the function.
- **Unsupervised learning** models a set of inputs, like clustering.
- **Semi-supervised learning** combines both labeled and unlabeled examples to generate an appropriate function or classifier.
- **Reinforcement learning** learns how to act given an observation of the world. Every action has some impact in the environment, and the environment provides feedback in the form of rewards that guides the learning algorithm.

- **Transduction** tries to predict new outputs based on training inputs, training outputs, and test inputs.

- **Learning to learn** learns its own inductive bias based on previous experience.

The next section describes the salient features of reinforcement learning.

2.6.1 Reinforcement Learning

Reinforcement learning is a sub-area of machine learning concerned with how an agent ought to take actions in an environment with the objective of maximizing some notion of a long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the agent ought to take in those states. As in most forms of machine learning, the agent must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics trial-and-error search and delayed reward are the two most important distinguishing features of reinforcement learning[45].

One of the differentiating challenges characteristic to reinforcement learning is the trade-off between exploration and exploitation. To maximize its reward, an agent must prefer effective actions tried in the past. However, to discover such actions, it has to experiment with untested actions. The agent has to exploit what it already knows in order to obtain a reward, but it also has to explore to make better action selections in the future.

In the reinforcement learning framework, the agent makes its decisions as a function of a signal from the environment called the environment's state. An environment is said to have the Markov property if its one-step dynamics enable predicting the next state and expected next reward given the current state and action[45]. A reinforcement learning task that satisfies the Markov property is called a Markov decision process or MDP. In reinforcement learning, the environment is typically formulated as a finite-state Markov decision process. Formally, the basic reinforcement learning model, as applied to MDPs, consists of:

- A set of environment states S

- A set of actions A and

- A set of scalar "rewards" in R

At each time t, the agent perceives its state $s_t \in S$ and the set of possible actions $A(s_t)$. It chooses an action $a \in A(s_t)$ and receives a new state s_{t+1} and a reward r_t from the environment. Based on these interactions, the agent must develop a policy $\pi : S \text{ x } T \to A$ (where T is the set of possible time indexes) which maximizes the quantity $R = r_0 + r_1 + \cdots + r_n$ for MDPs which have a terminal state, or the quantity $R = \sum_{t=0}^{\infty} \gamma^t r_t$ for MDPs without terminal states (where $0 \leq \gamma \leq 1$ is a future reward discounting factor)[8].

Almost all reinforcement learning algorithms are based on estimating value functions i.e. functions of states (or of state-action pairs) that estimate how good it is for the agent to be in a given state (or how good it is to perform a given action in a given state)[45]. Value function approaches attempt to find a policy that maximizes the return by maintaining a set of estimates of expected returns for one policy π (usually either the current or the optimal one). In such approaches, the goal is to estimate either the expected return starting from state s and following π thereafter, $V(s) = E[R|s, \pi]$ or the expected return when taking action a in state s and following π thereafter, $Q(s, a) = E[R|s, \pi, a]$.

According to [45], there are three fundamental classes of methods for solving the reinforcement learning problem namely dynamic programming, Monte Carlo methods, and temporal difference learning. Dynamic programming methods are well developed mathematically, but require a complete and accurate model of the environment, which is realistically almost impossible to obtain for any software system. Monte Carlo methods don't require a model and are conceptually simple, but are not suited for step-by-step incremental computation, rendering these methods unsuitable for self-adaptation as well. Temporal-difference methods require no model and are fully incremental, but are more complex to analyze, rendering these methods to be the most promising exploratory option for self-adaptive systems.

The next sections cover two popular off-policy and on-policy temporal difference algorithms namely Q-Learning and SARSA.

Q-Learning

Q-learning is a reinforcement learning technique that works by learning an action-value function that gives the expected utility of taking a given action in a given state and following a fixed policy thereafter. One of the strengths of Q-learning is that it is able to compare the expected utility of the available actions without requiring a model of the environment[7]. The problem model consists of an agent, states S and a number of actions per state A. By performing an action a, where $a \in A$, the agent can move from state to state. Each state provides the agent a reward or punishment. The goal of the agent is to maximize its total reward.

The algorithm therefore has a function which calculates the quality of a state-action combination:

$$Q : S \text{ x } A \to R \tag{2.1}$$

Before learning has started, Q returns a fixed initial value chosen by the implementer. Thereafter, each time the agent is given a reward, new values are calculated for each combination of a state s from S, and action a from A[7]. The core of the algorithm is a value iteration update. It assumes the old value and makes a correction based on the new information.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{learning rate}} \text{ x } [\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q(s_{t+1}, a)}_{\text{max future value}}}^{\text{expected discounted reward}} - \overbrace{Q(s_t, a_t)}^{\text{old value}}] \tag{2.2}$$

where

- r_{t+1} is the reward given at time $t + 1$
- $\alpha_t(s, a)$ is the learning rate such that $0 \leq \alpha_t(s, a) \leq 1$. The learning rate determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the most recent information.
- γ is the discount factor such that $0 \leq \gamma \leq 1$ The discount factor determines the importance of future rewards. A factor of 0 will make the agent "opportunistic" by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward. If the discount factor meets or exceeds 1, the Q values will diverge.

The above formula is equivalent to:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)(1 - \alpha_t(s_t, a_t)) + \alpha_t(s_t, a_t)[r_{t+1} + \gamma \max_a Q(s_{t+1}, a)] \tag{2.3}$$

The greatest advantage of the Q-Learning approach is its simplicity. It is easy to implement and demonstrates good results in applicable systems. However, the use of the Q-Learning algorithm is constrained. This is because the number of possible states of the environment and actions the agent can take must be finite. With increasing number of states and actions, storing the Q-value of each action morphs into a non-trivial task and has an adverse impact on overall performance and throughput. Additionally, problems with continuous state or action spaces cannot be solved by Q-Learning without using other techniques like the mapping of continuous spaces to a finite set of states or actions. Furthermore, if the agent always selects the action with the highest Q-Value for a given state, it will probably end up in a local maximum of its policy. This problem can be alleviated by an exploration strategy, which randomly makes the agent try actions it has not performed before.

SARSA

SARSA (State-Action-Reward-State-Action) is an on-policy algorithm for learning a Markov decision process policy, used in reinforcement learning. This name simply reflects the fact that the main function for updating the Q value depends on the current state of the agent s_t, the action the agent chooses a_t, the reward R the agent gets for choosing this action, the state s_{t+1} that the agent will now be in after taking that action, and finally the next action a_{t+1} the agent will choose in its new state. Taking every letter in the quintuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ yields the word SARSA.

Before learning has started, Q returns a fixed initial value chosen by the implementer. Thereafter, each time the agent is given a reward, new values are calculated for each combination of a state s from S, and action a from A[7]. The core of the algorithm is a value iteration update. It assumes the old value and makes a correction based on the new information.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(s_t, a_t)[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.4}$$

where

- r_{t+1} is the reward given at time $t + 1$
- $\alpha_t(s, a)$ is the learning rate such that $0 \leq \alpha_t(s, a) \leq 1$. The learning rate determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the most recent information.
- γ is the discount factor such that $0 \leq \gamma \leq 1$ The discount factor determines the importance of future rewards. A factor of 0 will make the agent "opportunistic" by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward. If the discount factor meets or exceeds 1, the Q values will diverge.

In the next section we describe applications of machine learning in self-adaptive systems.

2.7 Applications of Machine Learning in Self-Adaptive Systems

Several approaches based on machine learning have been proposed for managing systems performance in dynamic environments. The work in [13] proposes the use of reinforcement learning for influencing server allocation decisions in a multi-application data center environment. By observing the applications state, number of servers allocated to the application, and the reward specified by the service level agreement, a learning agent is then used to approximate $Q_\pi(s, a)$. To address poor scalability in large state spaces, the authors of [46] have proposed an approximation of the applications state by discretizing the mean arrival rate of page requests.

Recently, there have been proposals on how to adapt the use of policies to accommodate the reconfiguration of the managed environment. The authors of [16] and [17] propose an adaptive policy-based framework that supports dynamic policy configuration in response to changes within the managed environment. In their approach, policy adaptation describes the ability to modify network behavior by dynamically changing the policy parameters as well as selecting, enabling, or disabling policies at run-time. Reconfiguration events are used to trigger high-level control policies, which then determines which lower-level policies must be adapted to reconfigure the managed system.

The authors of [47] propose a framework which make use of reinforcement learning methodologies to perform adaptive reconfiguration of a distributed system based on tuning the coefficients of fuzzy rules. The focus is on the problem of dynamic resource allocation among multiple entities sharing a common set of resources. The paper demonstrates how utility functions for making dynamic resource allocation decisions, in stochastic dynamic environments with large state spaces, could be learned. The aim is to maximize the average utility per time step of the computing facility through the reassignment of resources (i.e. CPUs, memory, bandwidth, etc.) shared among several projects.

The research conducted in [18] proposes the use of reinforcement learning techniques in middlewares to improve and adapt the QoS management policy. In particular, a Dynamic Control of Behavior based on Learning (DCBL) Middleware is used to learn a policy that best fits the execution context. This is based on the estimation of the benefit of taking an action given a particular state, where the action, in this case, is a selection of a QoS level. It is assumed that, each managed application offer several operating modes from which to select, depending on the availability of resources.

The authors of [19] examine the effectiveness of reinforcement learning methodologies in determining how to use a set of active policies to meet different performance objectives. In order to do so, the model of the environment is learned on-line and used, at each timestep, to influence the policy guiding the agents interaction with the environment. To model systems dynamics from the use of an active set of policies, the authors use a mapping between the active policies and the managed systems states whose structure is derived from the metrics associated with the active policy conditions.

The authors of [6] have developed a decision-making framework (FUSION) for self-adaptive systems that combines feature-orientation, learning, and dynamic optimization. FUSION learns the impact of feature selection and feature interactions on the systems

Table 2.3: Applications of Machine Learning in Self-Adaptive Systems

Project Name	**Description**
[19]	A framework for adaptive policy-driven autonomic management using reinforcement learning methodologies.
FUSION[6]	A decision-making framework for self-adaptive systems that combines feature-orientation, supervised learning, and dynamic optimization.
[13]	Proposing the use of reinforcement learning for influencing server allocation decisions in a multi-application data center environment.
DCBL[18]	Proposing the use of reinforcement learning techniques in middlewares to improve and adapt the QoS management policy.
[16] and [17]	Proposing an adaptive policy-based framework that supports dynamic policy configuration in response to changes within the managed environment.
[47]	Proposing a framework that utilizes reinforcement learning methodologies to perform adaptive reconfiguration of a distributed system based on tuning the coefficients of fuzzy rules.

goals. It then uses this knowledge to efficiently adapt the system to satisfy as many user-defined goals as possible. FUSION uses a learning cycle to learn the impact of adaptation decisions in terms of feature selection on the systems goals. The authors have used the M5 model tree algorithm in their implementation. M5 is a supervised machine learning technique with the ability to eliminate insignificant features, it offers fast training and convergence, and efficient interaction detection. Using a prototype implementation of the system and a travel reservation system the authors have validated the approach and its properties and obtained promising results.

Table 2.3 summarizes the applications of machine learning in self-adaptive systems. The results obtained by all of the aforementioned research projects demonstrate that incorporating feedback control loops as first-class entities in the decision-making models of self-adaptive systems has significant potential. However, all of the aforementioned research has mostly been performed on systems with policy-based decision-making models, and thus cannot be used in systems with negotiable adaptation requirements. Moreover the feedback control loop realization mechanisms have been hard coded into the system and are not configurable. Furthermore, the systems can only satisfy one requirement at any given point in time, even if multiple requirements are being unsatisfied.

The next section summarizes the research gaps in the area of decision-making models in self-adaptive systems.

2.8 Research Gaps

The previous sections have highlighted different approaches used for realization of decision-making models in self-adaptive systems. In this section we summarize the existing research gaps in this area, which can be divided into the following categories:

- **Adaptation Requirements**: As mentioned before, the adaptation requirements of a system can be categorized as mandtory, negotiable and related negotiable requirements. To the best of our knowledge, none of the existing decision-making models have the ability to provide a comprehensive representation of all categories of adaptation requirements, particularly the related negotiable requirements. Consequently, none of the existing decision-making models can concurrently satisfy multiple adaptation requirements at any given point in time.

- **Conflict Detection and Resolution**: Conflicts can either exist in the adaptation requirements themselves or can exist in the actions required to satisfy multiple adaptation requirements. There is very limited support for detecting and resolving both types of conflicts in existing decision-making models.

- **Incorporation of Control Loops**: Since run-time uncertainity has become an inherent nature of software systems, it has been suggested that feedback control loops be incorporated as first-class entities in the decision-making model, so that the model can analyze and learn from the outcome of a previously executed decision. However, there are several challenges associated with this pertaining to the domain of modelling, verification and validation and software reengineering. Machine learning has been suggested as a realization mechanism for acheiving this. However, most of the projects have focussed on incorporating feedback control loops into policy-based decision-making models. To the best of our knowledge, very little work has been done on incorporating feedback control loops into goal-based decision-making models. Furthermore, all of the existing realization mechanisms have hard coded the leveraged machine learning technique, and thus have not exposed the control loop properties or made them configurable.

- **Provision for Specification of Configurable Algorithms**: Current state-of-the-art goal-oriented decision-making models have little or no support for specifying a *configurable voting algorithm* for choosing a winner requirement, if multiple requirements are unsatisfied at any given point in time. Moreover, none of the existing decision-making models that have incorporated feedback control loops as first-class entities have provided the ability to specify a *configurable machine learning algorithm* to analyze the effects of a previously executed action.

2.9 Summary

In this chapter research relevant to adpatation techniques employed by self-adaptive systems has been elaborated. We began the chapter by categorizing the different realization techniques used for decision-making models in self-adaptive systems. We then presented the advantages and disadvantages of each approach. Next, we explored a new and upcoming area of research in decision-making models namely the incorporation of feedback control loops as first-class entities in the model. We highligted the advantages of doing so, and also outlined some of the challenges associated with accomplishing this task. Next we explored the applications of machine learning in self-adaptive systems. In doing so, this chapter aimed to highlight some of the existing research gaps in each approach. In the next chapter we present the decision-making model developed in this dissertation which aims to provide a solution to some of the shortcomings that have been highlighted in this chapter.

Chapter 3

Proposed Decision Making Model

This chapter presents an overview of the decision making model developed in this thesis. The model aims to address the challenges of decision making in self-managing system development. We begin this chapter by describing the components of a decision-making model, followed by the requirements of a comprehensive and well-designed decision-making model. Next, we describe a goal-oriented decision-making model that can be used by self-managing systems and outline the advantages and disadvantages of this model. Subsequently, we describe the architecture of the decision-making model developed in this dissertation. We conclude this chapter by describing how a decision-making process can enforce the adaptation requirements by utilizing the proposed decision-making model.

3.1 Requirements and Constituent Components

In the self-adaptive software research community, there are discussions on the suitability of numerous models for adaptation requirements and domain information. These include architectural, configuration, performance and reliability models; to name a few[42]. The similarity in all these models is the common problem space concepts these models try to represent namely:

- **Goal Space**: Quality goals are commonly used to represent adaptation requirements. The role of quality goals in the adaptation manager is as important as their role in software engineering. Adaptation goals in the goal space are prescriptive statements that should be a mapping/translation of the adaptation requirements specified by the stakeholders. These goals capture different quality requirements for adaptable software. The satisfaction of these goals must be the primary focus of the adaptation manager.

- **Attribute Space**: For the monitoring and detecting processes, and in order to essentially trace adaptation goals, domain attributes should be captured and tracked. Attributes represent measurable and quantifiable properties of adaptable software. Attributes are system variables which may be controllable or non-controllable and are generated by the monitoring process from raw data collected from sensors.

- **Action Space**: Adaptation is based on changeability, and the adaptation manager needs effectors to apply changes to adaptable software. The ways an adaptation manager can apply changes are defined in the action space. Adaptation changes or actions basically aim to satisfy goals. Actions are the tasks that are performed by the system effectors. These actions can directly change self attributes. However, it is possible to impact context variables indirectly by some actions as well.

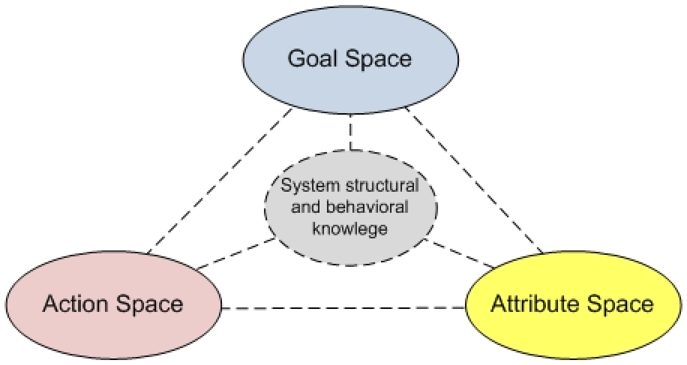


Figure 3.1: Adaptation Conceptual Model

These three concept spaces are the main tenets of the problem space. Figure 3.1 depicts the conceptual model combining these spaces. A fourth optional space can be also added to complement the model. This space may be required because some solutions need structural or behavioral information from the adaptable software system. The links between the main three spaces are shown by dotted lines, indicating that different patterns may bind the constituent entities of each space together, and all links do not exist in all solutions [42]. The initial step in defining the adaptation model is capturing entities in each concept space. These entities can be linked in each space together to build models (i.e. intraspace links).

In order to build a goal-oriented decision-making model, we need to first establish a mechanism to define and represent the goals, actions, attributes and their relationship in a goal-action-attribute-model. Additionally, we need to design a decision-making mechanism based on this model. The deciding processes will esstentially analyze this model to determine if any of the adaptation requirements are not being satisfied by the executing system i.e. if any of the system goals have been *activated*. If active goals exist in the system, the deciding processes will traverse this model to determine the actions that must be performed to restore normalcy to the sytem.

In the following section, we describe a goal-oriented decision making model that can be used by a self-managing system. We outline the pros and cons of this model and also leverage some of the pivotal features of this model in the development of our decision-making model.

3.2 GAAM - A Goal-Oriented Decision-Making Model

The Goal Action Attribute Model (GAAM) is a goal-oriented decision-making model developed by Salehie et al[44] that can be used in domain of self-adaptive systems. It defines a mechanism whereby the adaptation requirements are represented as goals, the data from the sensors belonging to the monitoring processes are represented as attributes, and the tasks that are performed by the effectors are represented as actions. GAAM represents and relates the adaptation goals, self/context attributes and adaptable actions in a well-defined structure which is used by the decision-making algorithm to select the best adaptable action to attain the adaptation requirements/system goals specified by the stakeholders.

Each entity in GAAM has several properties. Attributes represent measurable/quantifiable properties of adaptable software. Attributes can have different types like integer, time, etc. Moreover, each attribute belongs to a specific entity (e.g., method or component) or a level (e.g., system or subsystem).

Goals are represented using a hierarchy (high-level to low-level goals). Stakeholders often start to articulate high-level goals by specifying the desired behaviors of the system (e.g. performance and security). These goals are decomposed into low-level goals, which are more likely to be related directly to measurable attributes. Goals are stimulated by attributes, and the activation level determines whether a particular goal is activated or not. Activated goals are eligible to participate in the action selection mechanism. Goals also have priorities associated with them. The priority of a goal determines the weight of the goal in the GAAM model, and impacts its influence in the action selection mechanism. Priorities come from stakeholders opinions.

Adaptation actions are changes applicable to adaptable software entities using the provided effectors. These actions usually include some preconditions. Before considering an action eligible to be selected, its preconditions should be satisfied.

The approach employed by GAAM is summarized as follows:

Given a set of goals $GL = \{gl_i, i = 1 \ldots m\}$, a set of actions $AC = \{ac_i, i = 1 \ldots n\}$ and an attribute set $AT = \{at_i, i = 1 \ldots p\}$, GAAM solves the action selection problem

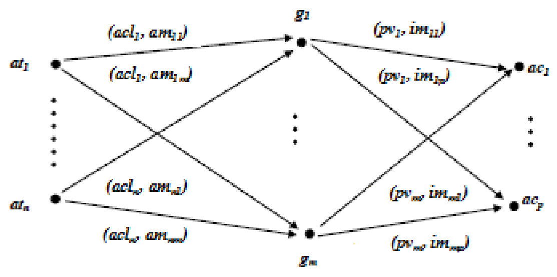


Figure 3.2: GAAM Graph

of choosing the most appropriate adaptation action ac_i to satisfy goals in different conditions of attributes. In order to do so, GAAM defines additional parameters to be associated with goals and actions. An activation level vector $ACL = \{acl_i, i = 1 \ldots m\}$ contains threshold values or a rules specifying how goals will be activated. A preference vector $PV = \{pv_i, i = 1 \ldots m\}$ contains preference values associated with goals as specified by the stakeholders. A precondition vector $PC = \{pc_i, i = 1 \ldots n\}$ contains the preconditions associated with adaptation actions. GAAM also defines matrices relating goals, actions and attributes. An impact matrix $IM = \{im_{ij}, i = 1 \ldots m, j = 1 \ldots p\}$ relates m goals to p actions. An activation matrix $AM = \{am_{ij}, i = 1 \ldots m, j = 1 \ldots n\}$ relates m goals to n attributes. An aspiration level matrix $ASL = \{asl_{ij}, i = 1 \ldots m, j = 1 \ldots n\}$ specifies the aspiration level of each goal in terms of n attributes. GAAM creates a graph $G = \{V, E\}$ in which the vertices are $V = \{GL \bigcup AT \bigcup AC\}$ and edges are $E = \{IM \bigcup AM \bigcup PV\}$. Figure 3.2 illustrates the graph developed and used by GAAM.

GAAM employs a goal-ensemble adaptation mechanism. The deciding process models a cooperative game based on the weighted voting of activated goals (i.e. the unsatisfied adaptation requirements). Here, it is assumed that the GAAM is continuously being traversed from attributes to actions for a specific adaptation period. Before making a decision, it is essential to determine which goals have been activated and which actions are feasible. The activated goals are voters, and feasible actions are eligible candidates.

Figure 3.3 illustrates the flow of the decision-making process employed by GAAM. The algorithm employed by the decision-making process is as follows. In the first step, all of the attributes are sensed and updated. In the next step, for each goal, the obtained attribute values are examined to determine if the values are outside their designated threshold limits. The next step then determines the goals that have been activated. In the next step, feasible actions are identified by checking the list of preconditions. In the next step, activated goals generate their preferred lists of actions (votes). The last step involves aggregating the votes, which is the actual voting mechanism. The action that receives the maximum votes i.e.

the “winner” action is the outcome of the analysis/traversal of the GAAM matrix, which must be executed by the system effectors in an attempt to restore normalcy to the system.

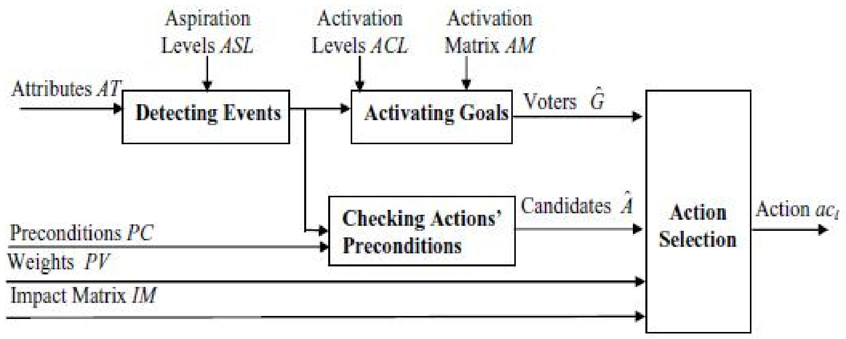


Figure 3.3: GAAM Action Selection Process

The ideas outlined in GAAM have been evaluated with a simulated MATLAB model of a news web site and the results obtained are promising. To the best of our knowledge, there is no known implementation of GAAM.

GAAM provides a goal-based model and decision-making mechanism to address action selection in self-adaptive software systems. It provides a mechanism whereby the the adaptation requirements are represented as first class entities in the self-adaptive system. Rule-based methods based on action policies implicitly utilize the goals to adapt the system. Compared to the aforementioned approach, the goal-based methods proposed in GAAM explicitly represent the goals, trace them at runtime, and include them in the deciding process. This characteristic leads to better traceability for administrators, and to more trust for stakeholders.

Moreover, GAAM facilitates multiple objective decision-making . The goal-based approach outlined in GAAM intrinsically supports multiple goals. Handling these goals in a rule-based method is not an easy task. This can result in a large set of rules with precisely adjusted priorities to address all the desired goals, while in GAAM, each goal is defined with its own utility for the systems adaptation actions. These goals are later coordinated in the action selection mechanism.

Furthermore, GAAM is a flexible decision-making model. Modifying rules at run-time is not always safe, and the systems behavior is not predictable. In the goal-based approach, the coordinator can handle the changes better. Goals, actions, and attributes can be added or removed from GAAM, although the authors have not made the claim that the system behavior is immune from any kind of change. The authors claim that the easiest way of changing GAAM is altering goal preferences.

However, GAAM does not provide a comprehensive mechanism to represent all the different types of adaptation requirements. GAAM views all adaptation requirements as negotiable requirements, and this may not be a viable solution for every self-adaptive system. Systems may have mandatory adaptation requirements that need to be satisfied under all circumstances. Hence a possible improvement to GAAM would be the ability to represent both mandatory and negotiable adaptation requirements.

Addtionally, the adaptation requirements of a system may be related, which implies that related requirements be grouped together and analyzed in a similar manner by the decision-making process. At present GAAM provides the ability to represent all the goals, actions and attributes in a single graph which is analyzed by the decision-making process. The concept of related goals, necessitates the concept of several graphs that are analyzed concurrently. Hence, an enhancement to GAAM would be the ability to represent similar negotiable requirements to form a *negotiable requirements cluster* and to support several such clusters of negotiable requirements. This allows the adaptation manager of an executing system to select mutilple unrelated goals, one from each cluster and thus satisfy multiple unrelated adaptation goals concurrently.

Furthermore, GAAM can be exended to support negotiable requirements augmented with a feedback control mechanism. This enables the adaptation manager to perform an evaluation on the effectiveness of a previously executed action, so that better decisions can be made in the future. This also provides a mechanism of modifying rules at run-time and evaluating the impact of the modification on the system's behaviour. The adaptation manger can utilize various on-policy reinforcement learning algorithms to evaluate the effectiveness of an executed action. The negotiable adaptive requirements that support this functionality can be referred to as *adaptive negotiable requirements*.

The next section outlines the functional requirements of the decision-making model developed in this dissertation that leverages the fundamentals of GAAM and extends it to support the aforementioned features.

3.3 Decision-Making Model Requirements

Based on the aforementioned discussion, a well-designed and comprehensive goal-oriented decision-making model must satisfy the following requirements:

- **FR1: Representation of system attributes** The decision-making model must provide a comprehensive representation of the attributes of the adaptable software and its execution context i.e. it must provide the ability to represent data produced by the sensors.

- **FR2: Representation of adaptation actions** The decision-making model must provide a means to represent the actions that can be performed on the adaptable software by the effectors.

- **FR3: Representation of adaptation requirements as goals** The decision-making model must provide a representation of all the adaptation requirements i.e. the stakeholders' expectations of the system as goals.

- **FR4: Distinguishable representation of mandatory and negotiable goals** The decision-making model must provide a clearly distinguishable representation of mandatory and negotiable adaptation requirements.

- **FR5: Support for negotiable goal clusters** The decision-making model must provide the ability to represent similar negotiable adaptation requirements as a cluster, so that they can be analyzed by the adaptation manager as a single entity to produce a "winner" requirement. It must also provide the ability to support several such negotiable clusters.

- **FR6: Support for pluggable voting algorithms for negotiable goal clusters** The decsion-making model must provide the adaptation manager with the ability to use a variety of voting algorithms while choosing a winner goal from the negotiable goal cluster.

- **FR7: Support for adaptive negotiable goal clusters** The decision-making model must support the ability to provide negotiable clusters with optional feedback control loops. This facilitates evaluation of the effectiveness of a previously executed effector action by the adaptation manager.

- **FR8: Support for pluggable reinforcement learning algorithms for adaptive negotiable goal clusters** The decision-making model must provide the adaptation manager with the ability to use a variety of on-policy reinforecement learning algorithms to evaluate the effectiveness of a previously executed effector action for adaptive negotiable goal clusters.

- **FR9: Representation of goal-attribute relationship** The decision-making model must provide a representation of the relationship between an adaptation requirement and the relevant data produced by the sensors. Note that this can be a many-to-many relationship.

- **FR10: Representation of goal-action relationship** The decision-making model must provide a representation of the relationship between an adaptation requirement and the relevant actions that can be performed by the effectors. Note that this can also be a many-to-many relationship.

The next section outlines the architecture of the decision-making model developed in this dissertation.

3.4 Proposed Decision-Making Model Architecture

Figure 3.4 shows the architecture of the decision-making model. Table 3.1 provides a summary of all the configurable entities of the decision-making model and the functional requirement satisfied by each entity. The **problem space** of the proposed decision-making model is composed of the following:

- **Attribute Space**: For the monitoring and detecting processes, and in order to essentially trace adaptation goals, the measurable properties of the adaptable software and its context should be captured and tracked. The attribute space is the superset of all such properties.
- **Action Space**: Adaptation changes basically aim to satisfy goals. The ways an adaptation manager can apply changes are defined in the action space.
- **Goal Space**: The goal space contains prescriptive statements that should be a mapping/translation of the adaptation requirements specified by the stakeholders.

Each of the aforementioned spaces is composed of several entities and sub-entities. Additionally, there are certain entities that belong to both the goal space and the action space; and some that belong to both the goal space and the attribute space. Subsequent sections will provides additional details about each of these entities and their comprising sub-entities.

3.4.1 Attribute Space

For the monitoring and detecting processes, and in order to essentially trace adaptation goals, the measurable properties of the adaptable software and its context should be captured and tracked. The attribute space is the superset of all such properties. The attribute space is composed of one or more *attribute*(s).

Attribute

Attributes represent measurable and quantifiable properties of adaptable software. Attributes are system variables which may be controllable or non-controllable and are generated by the monitoring process from raw data collected from sensors. In order to adequately model a self or context attribute, the following information must be provided:

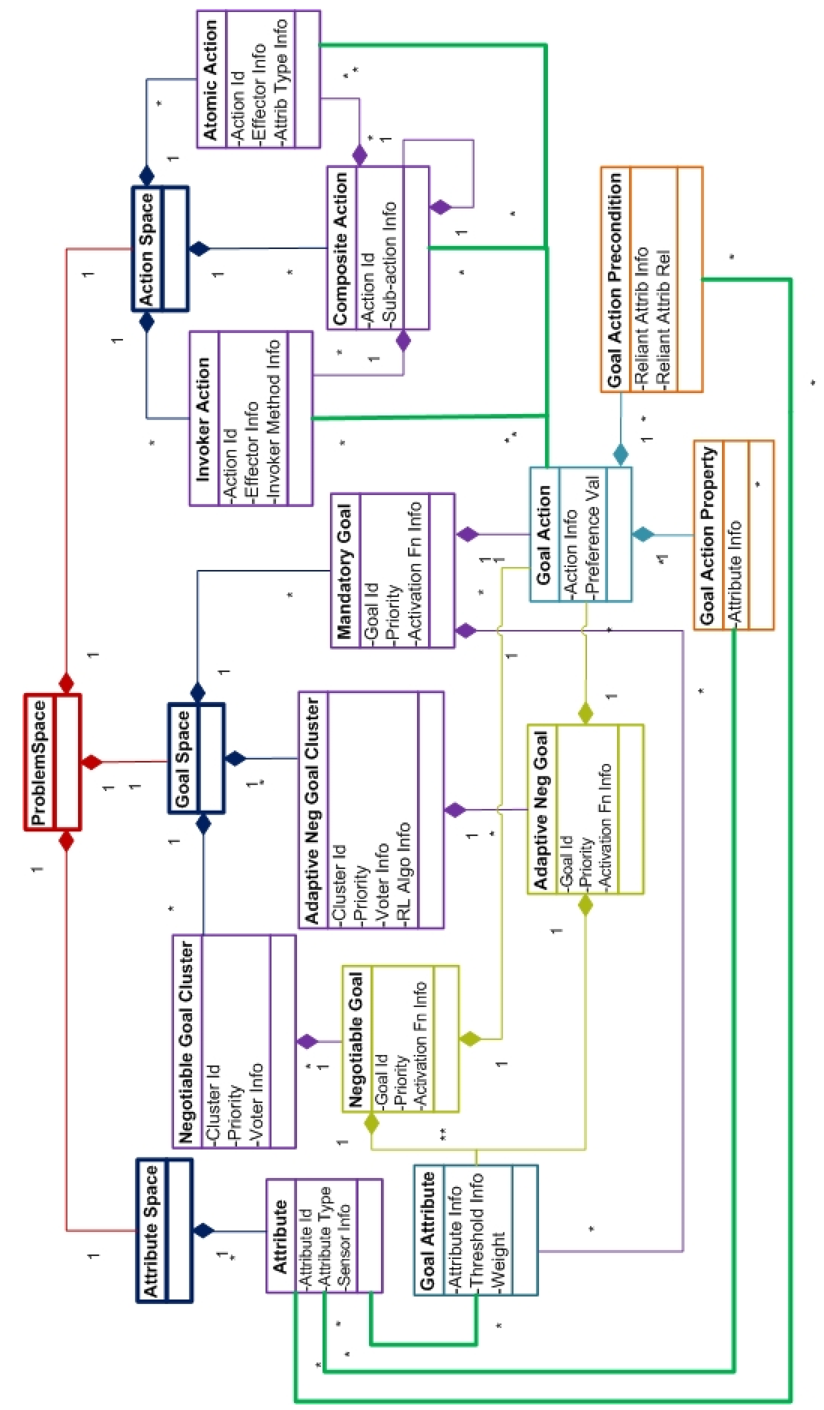


Figure 3.4: Proposed Decision-Making Model Architecture

Table 3.1: Configurable Entities of the Proposed Decision-Making Model

Entity	**Description**	**Functional Requirement**
Attribute	Attributes represent measurable and quantifiable properties of adaptable software.	FR1
Atomic Action	Atomic actions involve executing any setter method by the effectors.	FR2
Invoker Action	Invoker actions involve executing a method exposed by the adaptable software, which is accessible to the effectors.	FR2
Composite Action	Composite actions are composed of a series of invoker, atomic actions and composite actions; and involve sequential execution of each action.	FR2
Mandatory Goal	Mandatory goals represent the mandatory adaptation requirements that must be satisfied by the system under all circumstances.	FR3, FR4
Negotiable Goal Cluster	Negotiable goal clusters represent groups of flexible adaptation requirements.	FR3, FR4, FR5, FR6
Negotiable Goal	Negotiable goals are contained within negotiable goal clusters and are not top-level entities of the model.	FR3, FR4
Adaptive Negotiable Goal Cluster	Adaptive negotiable goal clusters represent negotiable adaptation requirements that have been modeled with a control loop to facilitate evaluation of the effectiveness of a previously executed action.	FR3, FR7, FR8
Adaptive Negotiable Goal	Adaptive negotiable goals are contained within adaptive negotiable goal clusters and are not top-level entities of the model.	FR3, FR7
Goal Attribute	Goal attributes represent the relationship between an attribute and a goal.	FR9
Goal Action	Goal actions represent the relationship between goals and actions.	FR10
Goal Action Property	Goal action properties represent the additional properties that must be specified while invoking a particular action.	FR10
Goal Action Precondition	Goal action preconditions represent the preconditions that must be satisfied before an action can be invoked.	FR10

- **Attribute Id**: An identifier by which the attribute can be uniquely identified in the model.
- **Attribute Type**: The data type of the attribute. The data type can be one of the primitive data types or a complex data type.
- **Sensor Info**: The information about the sensor which produces the data for this attribute. e.g. If the sensors of Java-based system are implemented as MBeans, the sensor information includes the specifying MBean connection information and the name of the MBean *getter* method that must be invoked to obtain the data for this attribute.

3.4.2 Action Space

Adaptation is based on changeability, and the adaptation manager needs effectors to apply changes to adaptable software. The ways an adaptation manager can apply changes are defined in the action space. Adaptation changes or actions basically aim to satisfy goals. Actions are the tasks that are performed by the system effectors. These actions can directly change self attributes. However, it is possible to impact context variables indirectly by some actions as well. Actions can be further classified as *invoker*, *atomic* and *composite* actions. Each action category has certain properties that must be specified while modeling the action type.

Invoker Action

Invoker actions involve executing a method exposed by the adaptable sofware which is accessible to the effectors to restore normalcy to the system. In order to adatequately model invoker actions, the following information must be provided:

- **Action Id**: An identifier by which the action can be uniquely identified in the model.
- **Effector Info**: The information about the effector which invokes the method for this action. e.g. If the sensors of Java-based system are implemented as MBeans, the effector information includes the specifying MBean connection information.
- **Invoker Method Info**: Complete information about the method that the effector must invoke. e.g. In the case of MBeans, this information includes specifying the method name and the complete method signature.

Since specifying the complete method signature involves a deep understanding of a programming languages like Java, and more often than not, actions involve invoking *setter* methods exposed by the adaptable software, our model provides an optimized version of invoker actions called atomic actions.

Atomic Action

Atomic actions involve executing any one *setter* method by the effectors to restore normalcy to the system. In order to adatequately model atomic actions, the following information must be provided:

- **Action Id**: An identifier by which the action can be uniquely identified in the model.
- **Effector Info**: The information about the effector which invokes the *setter* method for this action.
- **Attrib Type Info**: The information about the data type of the attribute that is passed as a parameter into the *setter* method that the effector must invoke.

Composite Action

In certain cases, a system may require the sequential execution of several tasks in order to restore normalcy to the system. In order to support this functionality, our model includes an entity called a composite action. Composite actions are composed of a series of invoker,atomic or composite actions and involve sequential execution of each action. In order to adequately model composite actions, the following information must be provided:

- **Action Id**: An identifier by which the action can be uniquely identified in the model.
- **Sub-action Id**: Identifier(s) by which the comprising actions can be uniquely identified in the model. This list cannot include the identifier of the composing composite action.

3.4.3 Goal-Attribute Space

Although the goal-attribute space has not been explicitly listed in figure 3.4, it is a conceptual space that contains entities belonging to both the goal space and the attribute space. It is composed of one or more *goal attribute*(s).

Goal Attribute

Since an attribute can be associated with multiple goals, and each association may have unique characteristics, the model has been equipped with an entity called a goal attribute. Goal attributes represent the relationship between an attribute and a goal. Goal attributes can only exist as sub-entities within a goal. In order to adequately model goal attributes, the following information must be provided:

- **Attribute Info**: The identifier of the attribute.
- **Threshold Info**: The threshold that the attribute value must stay within. If the attribute value falls outside the designated threshold, then the containing goal may be activated. The threshold value specification can vary from specifying a minimum and maximum value, a particular value, a null value or some other complex representation; and the model must provide the ability to represent this. Hence support for specifying a minimum and/or maximum value, a particular value and a null value has been built into the model. For complex threshold value specification, the model leverages the *adapter design pattern* and provides the ability to specify a custom Java class that must implement a pre-defined interface.
- **Weight**: The influence of this attribute on the goal. The sum of the weights of all goal attributes defined in a cluster must equal 1 or a multiple of 10. This property is primarily used if this goal attribute belongs to a goal belonging to an adaptive negotiable cluster. For further details, please refer to section 3.5.1.

3.4.4 Goal-Action Space

Although the goal-action space has not been explicitly listed in figure 3.4, it is a conceptual space that contains entities belonging to both the goal space and the action space. It is composed of one or more *goal action*(s).

Goal Action

Goal actions represent the relationship between goals and actions. Since an action can be associated with multiple goals, and each association may have unique characteristics, the model has been equipped with an entity called a goal action. In order to adequately model goal actions, the following information must be provided:

- **Action Info**: The identifier of the action.

- **Preference Value**: The preference value of the action i.e. the amount by which the composing goal, prefers this action over another action associated with the goal.

Goal actions are further comprised of *goal action precondition*(s) and *goal action property*(ies).

Goal Action Precondition

Goal action preconditions represent the preconditions that must be saitsfied before an action can be invoked. Since an action can be associated with multiple goals, it may require different preconditions for each goal. Hence, the preconditions are associated with goal actions as opposed to the action definition itself. In order to adequately model a goal action precondition, the following information must be provided:

- **Reliant Attrib Info**: In order to evaluate the precondition of an action, access may be required to the values of some attributes defined in the system. This property contains the information about the identfiiers and values of the attributes that must be provided to determine if the precondition of the goal action has been satisfied.
- **Reliant Attrib Rel**: In addition to specifying information about the attributes, the precondition evaluator also needs information about the relationship that must exist within the attributes. The relationship can range from simple relationship (i.e. determining if two values are equal, greater than, less than, not equal etc. to each other) or a complex relationship; and the model must provide the ability to represent this. Hence support for specifying a simple relationship has been built into the model. For complex relationships, the model leverages the *adapter design pattern* and provides the ability to specify a custom Java class that must implement a pre-defined interface.

Goal Action Property

Goal action properties represent the additional properties that must be specified while invoking a particular action. In order to adequately model goal action properties, the following information must be provided:

- **Attribute Info**: Information about the attribute value that must be set by the atomic action invoking a setter method, or the invoker action invoking any method. For complex attribute values, the model leverages the *adapter design pattern* and provides the ability to specify a custom Java class that must implement a pre-defined interface. For composite actions, the properties for each constituent action must be provided in sequence.

3.4.5 Goal Space

Adaptation goals in the goal space are prescriptive statements that should be a mapping/translation of the adaptation requirements specified by the stakeholders. These goals capture different quality requirements for adaptable software. As a result, the goal space is a superset of all the categories of adaptation requirements and hence is composed of *mandatory goal*(s), *negotiable goal cluster*(s) and *adaptive negotiable goal cluster*(s).

Mandatory Goals

Mandatory goals represent the mandatory adaptation requirements that must be satisfied by the system under all circumstances. These goals contain goal attributes and goal actions as sub-entities. Since mandatory goals can be viewed as a representation of action policies, only one goal action can be associated with it. In order to adequately model mandatory goals, the following information must be provided:

- **Goal Id**: An identifier by which a goal can be uniquely identified in the model.
- **Activation Fn Info**: When the adaptation requirements of a system are not satisfied, the goal representing the requirement is said to be *triggered*. This occurs when the attributes associated with the goal fall outside their designated threshold limits and are *activated*. Activation functions represent the relationship that the activated goal attributes must satisfy in order to trigger this goal. The relationship between the attributes can range from a simple boolean relationship to a complex mathematical relationship and the model must provide the ability to accommodate this. Hence, the common boolean relationships have been directly built into the model. For complex relationships, the model leverages the *adapter design pattern* and provides the ability to specify a custom Java class that must implement a pre-defined interface.
- **Priority**: The priority associated with this goal. This is used for conflict detection and resolution. For further details, please refer to section 3.5.3.

Negotiable Goal Clusters

Negotiable goal clusters represent groups of related negotiable adaptation requirements that allow trade-off analysis in the decision-making process. These clusters contain one or more *negotiable goal*(s). If a cluster contains more than one active goal at any given point in time, only one winner goal is selected from the cluster. Moreover, only one of the goal actions associated with the winner goal will be invoked on the adaptable software. In order to adequately model negotiable goal clusters, the following information must be provided:

- **Cluster Id**: An identifier by which the goal cluster can be uniquely identified in the model.
- **Voter Info**: The information about the voting algorithm used to determine a winner goal. The voting algorithm can range from a simple algorithm (e.g. borda count, winner takes all) to a complex algorithm; and the model must provide the ability to support this. Hence, the simple voting algorithms have been built into the model. For complex algorithms, the model leverages the *adapter design pattern* and provides the ability to specify a custom Java class that must implement a pre-defined interface.
- **Priority**: The priority associated with this cluster. This is used for conflict detection and resolution. For further details, please refer to section 3.5.3.

Negotiable Goal

Negotiable goals are contained as sub-entities within negotiable goal clusters. Negotiable goals contain goal attributes and goal actions as sub-entities. However, unlike mandatory goals, several goal actions can be associated with one negotiable goal. Negotiable goals assign preference values for each goal action. The action selection process determines the goal action to be executed for an activated negotiable goal. In order to adequately model negotiable goals, the following information must be provided:

- **Goal Id**: An identifier by which a goal can be uniquely identified in the model.
- **Activation Fn Info**: When the adaptation requirements of a system are not satisfied, the goal representing the requirement is said to be *triggered*. This occurs when the attributes associated with the goal fall outside their designated threshold limits and are *activated*. Activation functions represent the relationship that the activated goal attributes must satisfy in order to trigger this goal. The relationship between the goal attributes must either be a boolean *and* or a boolean textitor relationship.
- **Priority**: The priority associated with this goal. This is used in the action selection process whereby a winner goal action is selected for the cluster. For further details, please refer to section 3.5.2.

Adaptive Negotiable Goal Clusters

Adaptive negotiable goal clusters incorporate feedback control loops as first-class entities in the decision-making process. The realization methodology used for feedback control loops is reinforcement learning. The incorporation of feedback control loops facilitates the evaluation of the effectiveness of a previously executed action, so that better and more informed decisions can be made in the future. These clusters contain one or more *adaptive neg goal*(s). In order to adequately model adaptive negotiable goal clusters, the following information must be provided:

- **Cluster Id**: An identifier by which the goal cluster can be uniquely identified in the model.
- **Voter Info**: The information about the voting algorithm used to determine a winner goal.
- **Priority**: The priority associated with this cluster. This is used for conflict detection and resolution. For further details, please refer to section 3.5.3.
- **RL Algo Info**: The reinforcement learning algorithm used to evaluate the effectiveness of an executed action. The reinforcement learning algorithm can range from Q-Learning and SARSA to a customizable reinforcement learning algorithm. To enable specification of a custom reinforcement learning algorithm, the model leverages the *adapter design pattern.* It provides the ability to specify a custom Java class that must implement a pre-defined interface.

Adaptive Neg Goal

Adaptive negotiable goals are contained as sub-entities within adaptive negotiable goal clusters. In order to adequately model adaptive negotiable goals, the following information must be provided:

- **Goal Id**: An identifier by which a goal can be uniquely identified in the model.
- **Activation Fn Info**: Activation functions represent the relationship that the activated goal attributes must satisfy in order to trigger this goal.
- **Priority**: The priority associated with this goal. This is used in the action selection process whereby a winner goal action is selected for the cluster. For further details, please refer to section 3.5.2.

The next section describes how the decision-making model is used to ensure compliance of the adaptation requirements.

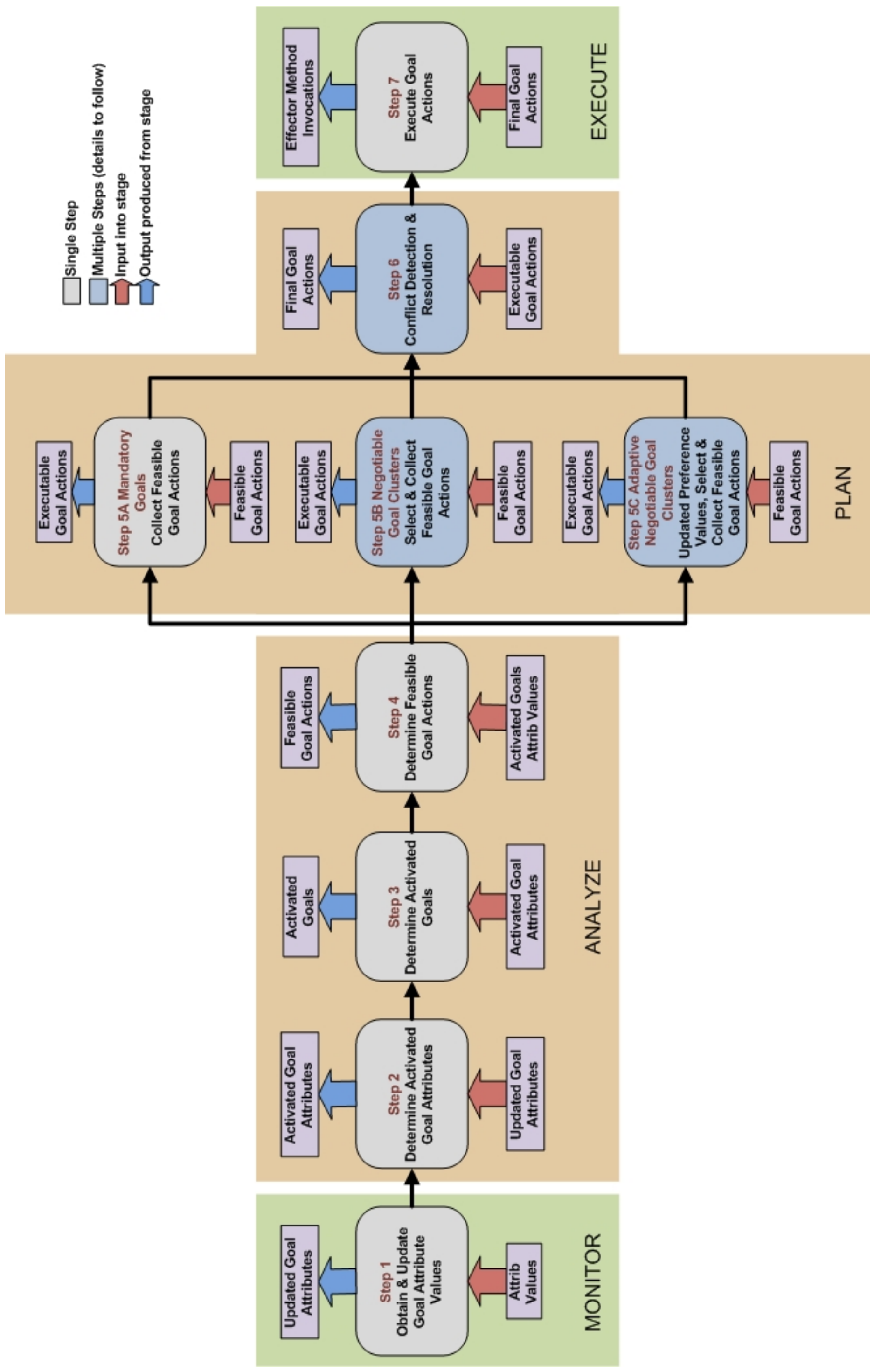

MONITOR
Step 1
Obtain & Update Goal Attribute Values
Attrib Values
Updated Goal Attributes
ANALYZE
Step 2
Determine Activated Goal Attributes
Updated Goal Attributes
Activated Goal Attributes
Step 3
Determine Activated Goals
Activated Goal Attributes
Activated Goals
Step 4
Determine Feasible Goal Actions
Activated Goals Attrib Values
Feasible Goal Actions
PLAN
Step 5A Mandatory Goals
Collect Feasible Goal Actions
Feasible Goal Actions
Executable Goal Actions
Step 5B Negotiable Goal Clusters
Select & Collect Feasible Goal Actions
Feasible Goal Actions
Executable Goal Actions
Step 5C Adaptive Negotiable Goal Clusters
Updated Preference Values, Select & Collect Feasible Goal Actions
Feasible Goal Actions
Executable Goal Actions
Step 6
Conflict Detection & Resolution
Executable Goal Actions
Final Goal Actions
EXECUTE
Step 7
Execute Goal Actions
Final Goal Actions
Effector Method Invocations
Single Step
Multiple Steps (details to follow)
Input into stage
Output produced from stage

Figure 3.5: Adaptation Process Flow using Proposed Decision-Making Model

3.5 Adaptation Process Flow using Proposed Decision-Making Model

Figure 3.5 describes the procedure employed by the decision-making process to ensure that the adaptation requirements are being satisfied by the system. It constis of the following steps:

- **Step 1: Obtain and Update Goal Attribute Values**: In this step, the values of all attributes defined in the decision-making model are obtained from the sensors via the adaptation framework. Subsequently, the values of the goal attributes that refer to these attributes are updated.
- **Step 2: Determine Activated Goal Attributes**: In this step, the thresholds of all the goal attributes are evaluated to determine if the system is operating within its designated limits. If the goal attribute value is outside its threshold limits, the goal attribute is deemed to be *activated*.
- **Step 3: Determine Activated Goals**: In this step, the activation functions associated with all the goals with activated goal attributes are evaluated to determine if the goal is currently being satisfied by the system. If the evaluation of the activation function determines that the goal is not being satisfied, then the goal is considered to be *activated*.
- **Step 4: Determine Feasible Goal Actions**: In this step, the preconditions of all the goal actions belonging to the activated goals are evaluated to obtain the list of feasible goal actions.
- **Step 5: Select and Collect Executable Goal Actions**: In this step, the list of feasible goal actions is evaluated by the goals/goal clusters to obtain the list of executable goal actions. The evaluation procedure employed depends on the type of entity.
  - **Step 5A: Mandatory Goals**: Since mandatory goals represent adaptation requirements that must be satisfied under all cirumstances, the goal actions belonging to mandatory goals need no further processing and are directly added to the list of executable goal actions.
  - **Step 5B: Negotiable Goal Clusters**: Negotiable goal clusters execute an *action selection process* on the list of feasible goal actions per goal cluster, to obtain a “winner” goal action. Details about the action selection process are provided in section 3.5.2. One winner action is selected per goal cluster and is added to the list of executable goal actions.

  - **Step 5C: Adaptive Negotiable Goal Clusters**: Adaptive negotiable goal clusters utilize the feedback control loop to update the preference value of each feasible goal actions. For further details on how this is accomplished, please refer to section 3.5.1. After updating the goal action preference values, the feasible goals undergo an action selection process to select a "winner" goal action per goal cluster. Details about the action selection process are provided in section 3.5.2. The winner goal action is then added to the list of executable goal actions.

- **Step 7: Conflict Detection and Resolution**: As mentioned before, two categories of conflicts can exist in a decision-making model. Conflicts pertaining to the adaptation requirements i.e. the goal definitions and conflicts pertaining to the actions to be executed for active goals. The proposed decision-making model is equipped to process the latter category of conflicts. In this step, the list of executable goal actions is scanned to determine if the goal actions to be executed are in conflict. If conflicts are detected, the conflicts are resolved to obtain the list of final goal actions. Details about the employed conflict detection and resolution mechanism can be obtained in section 3.5.3.

- **Step 8: Execute Selected Goal Actions**: In this step, all of the goal actions belonging to the list of final goal actions are executed concurrently by invoking the appropriate effector methods via the adaptation framework.

The next section describes how feedback control loops have been incorporated as first-class entities in the decision-making process.

3.5.1 Incorporation of Control Loops in the Decision-Making Process

In order to leverage reinforcement learning as a realization mechanism for control loops in adaptive negotiable clusters, we need to build a model of the adaptable software. A model, in reinforcement learning, describes any feedback that guides the interaction between the learning agent and its environment. This interaction is driven by the choices of actions and the behavior of the system as a consequence of taking those actions. i.e. a model in a sense can be viewed as a state-transition graph composed of states and transitions. The outcome of the actions executed by a learning agent may transition a system from an unfavorable state (i.e. when the adaptation requirements are not being satisfied) to a more favorable state (i.e. when the adaptation requirements are being fully/partially satisfied) and vice-versa. The agent obtains a postive reward upon encountering a favorable state and receives a negative reward upon an encounter with an unfavorable state. The agent learns by maximizing its positive reward count.

Each adaptive negotiable cluster needs to build a model of the adaptable software and its context. Consider an adaptive negotiable cluster with a set of m goals $GL = \{gl_i, i = 1 \ldots m\}$, a set n of goal actions $AC = \{ac_i, i = 1 \ldots n\}$ representing the goal actions associated with all the goals and a set of p unique goal attributes $AT = \{at_i, i = 1 \ldots p\}$ representing the unique goal attributes all the goals in the cluster. Since the acitvated goal attributes affect the triggering of a goal, the unique goal attributes can be used to represent the states of the adaptable software model. Each unique goal attribute $at_i, i = 1 \ldots p$ can be represented by the binary values $\{1, 0\}$ where 1 represents an activated goal attribute and 0 represents an unactivated goal attribute. Thus the p goal attributes in the cluster can be represented by a bitmap of size $2^p - 1$, where each bitmap entry represents a state in the model. We subtract 1 from 2^p since it is guaranteed that at least one of the states encountered will be a favorable state, and the decision-making model can be optimized to only keep track of the unfavorable states.

In order to evaluate the preference value of an executed goal action $ac_i, i = 1 \ldots n$, we need to keep track of the system state encountered, the goal action executed, the outcome of the action (which can be obtained in the next monitoring iteration) and then update the preference value of the goal action associated with this state based on a positive/negative outcome. This implies that each goal action may be associated with a bitmap of size $2^p - 1$. Since these maps are stored in memory, using this approach may become impractical for systems which are constrained by memory usage. Consequently, we do not allocate all of this memory at startup and build the model of the system dynamically as each state is encountered and a particular goal action is selected for execution. Our belief is that all states will not be encountered by every goal action in the system, and not all $2^p - 1$ states will be unfavorable states.

Figure 3.5.1 describes the procedure followed by a decision-making process utilizing the proposed decision-making model to update the goal action preference values. The procedure can be outlined as follows:

- **Step 1: Determine System State**: In this step, the goal attribute values are used to determine the system state. The system state is represented by a bitmap of size 2^p (if it contains p unique goal attributes). If a goal attribute is activate, the relevant bit is set to 1, otherwise it is set to 0.

- **Step 2: Propagate Variables to Custom RL Algorithm**: In this step, the bitmap representing the current system state, the previous system state, information about the previous winner goal action, previous activated goals and feasible goal actions and current feasible goal actions is propagated to the configurable reinforcement learning algorithm.

- **Step 3: Custom RL Algorithm Processing**: In this step, the configurable reinforcement learning algorithm analyzes the current and previous system state and uses

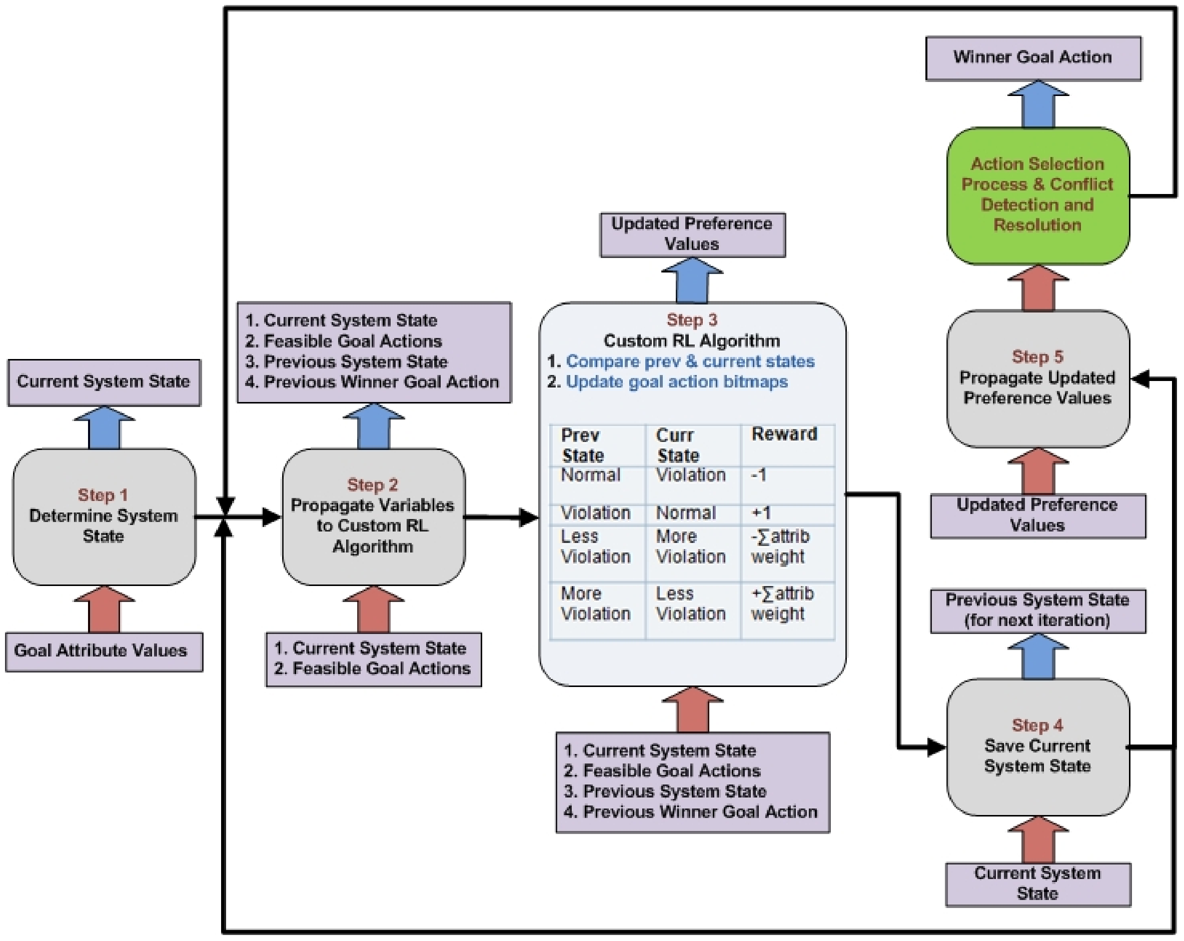

Winner Goal Action
Action Selection Process & Conflict Detection and Resolution
Updated Preference Values
1. Current System State
2. Feasible Goal Actions
3. Previous System State
4. Previous Winner Goal Action
Current System State
Step 1
Determine System State
Goal Attribute Values
Step 2
Propagate Variables to Custom RL Algorithm
1. Current System State
2. Feasible Goal Actions
Step 3
Custom RL Algorithm
1. Compare prev & current states
2. Update goal action bitmaps
Prev State | Curr State | Reward
Normal | Violation | -1
Violation | Normal | +1
Less Violation | More Violation | -∑attrib weight
More Violation | Less Violation | +∑attrib weight
1. Current System State
2. Feasible Goal Actions
3. Previous System State
4. Previous Winner Goal Action
Step 5
Propagate Updated Preference Values
Updated Preference Values
Previous System State (for next iteration)
Step 4
Save Current System State
Current System State

Figure 3.6: Using Control Loops to Update Goal Action Preference Values

this information to update the preference values of the goal actions. The processing is algorithm specific. However, the general premise is as follows:

  – If the system had no active goals in the previous state, and has active goals in the current state, then the preference value of the previous winner goal action is decremented by 1.
  – If the system had active goals in the previous state, and does not contain any active goals in the current state, then the preference value of the previous winner goal action is incremented by 1.
  – If the previous state had less active goals than the current state, the preference value of the previous winner goal action is decremented by an amount that equals the sum of all the goal attributes that are active in the current state, and were not active in the previous state.
  – If the previous state had more active goals than the current state, the preference value of the previous winner goal action is incremented by an amount that equals the sum of all the goal attributes that were active in the previous state, and are not active in the current state.

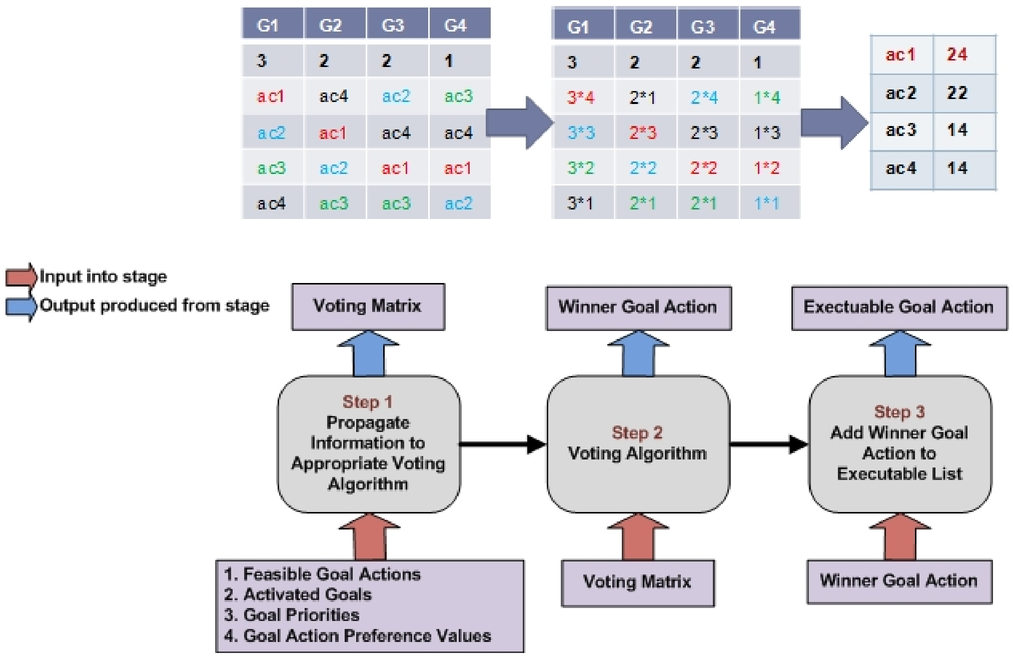
  As mentioned before, the model has two built-in reinforcement learning algorithms namely Q-Learning and SARSA, but an end-user can use any reinforcement learning algorithm provided it implements a specified interface.

- **Step 4: Save Current System State**: In this step, the current system state, current active goals and current feasible goal actions are saved to be used as “previous values” in the next iteration.
- **Step 5: Propagate Updated Preference Values**: In this step, the updated preference values are propagated to the action selection process and the conflict detection and resolution stages. The output of the conflict detection and resolution stage, i.e. the final goal actions are propagated back to the control loop to be used as “previous values” in the next iteration.

The next section describes the action selection process employed by negotiable and adaptive negotiable goal clusters to select a winner goal action.

3.5.2 Action Selection Process used by Goal Clusters

Figure 3.5.2 describes the procedure followed by the action selection process utilized by the goal clusters to select a winner action if a cluster contains multiple feasible actions. This procdedure is employed by both negotiable and adaptive negotiable goal clusters. As



G1	G2	G3	G4
3	2	2	1
ac1	ac4	ac2	ac3
ac2	ac1	ac4	ac4
ac3	ac2	ac1	ac1
ac4	ac3	ac3	ac2

G1	G2	G3	G4
3	2	2	1
3*4	2*1	2*4	1*4
3*3	2*3	2*3	1*3
3*2	2*2	2*2	1*2
3*1	2*1	2*1	1*1

ac1	24
ac2	22
ac3	14
ac4	14

Figure 3.7: Action Selection Process used by Goal Clusters

mentioned before, each goal has a priority and each goal action has a preference value assigned to it. These priorities and preference values are set by the stakeholders, and play a major role in the action selection process. The goal priorities are always static. The goal action preference values are static for negotiable goal clusters and are modified by the control loop for adaptive negotiable goal clusters. Using another control loop to modify the priorities of the goals in adaptive negotiable goal clusters is outside the scope of this thesis, and can be viewed as a potential future research topic. The action selection process can be outlined as follows:

- **Step 1: Propagate Information to Appropriate Voting Algorithm**: In this step, the activated goals, goal prirorities, feasible actions and feasible action preference values are transformed into a matrix-like format and propagated to a configurable voting algorithm for further processing.

- **Step 2: Voting Algorithm**: In this step, the voting algorithm evaluates its input to select a winner action. The processing is algorithm specific. However, if *borda-count* is used, the processing will be as follows: Consider 4 active goals, G_1, G_2, G_3, G_4 with priorties of $3, 2, 2, 1$ respectively. Each goal has 4 feasible goal actions namely ac_1, ac_2, ac_3, ac_4 with preference values as shown in figure 3.5.2 where the preference values are $4, 3, 2, 1$ for goal G_1 and so on. Borda-count evaluates the weight of each action as the sum of the product of the goal priority and is preference value for each goal, and selects action ac_1 as the winner goal action.

- **Step 3: Add Winner Goal Action to Executable List**: In this step, the winner goal actions are added to the list of executable goal actions.

The next section describes the conflict detection and resolution mechanism employed by a decision-making process using the prposed decision-making model.

3.5.3 Conflict Detection and Resolution

Two categories of conflicts can exist in a decision-making model. Conflicts pertaining to the adaptation requirements i.e. the goal definitions and conflicts pertaining to the actions to be executed for active goals. The proposed decision-making model is equipped to process the latter category of conflicts. In this step, the list of executable goal actions is scanned to determine if the goal actions to be executed are in conflict. If multiple goals and/or goal clusters select the same goal action, with different execution properties, the goal actions are deemed to be in conflict.

As mentioned before, the decision-making model has provided the ability to specify priorities for mandatory goals and goal clusters which are hard coded and set by the stake holders. These priorities are used to resolve conflicts between goal actions as outlined below:

- **Mandatory Goals take Precendence**: Conflicts between goal actions belonging to mandatory goals and goal clusters are resolved based on the fact that mandatory goals take precedence regardless of priority. This is because these goals represent adaptation requirements that must be satisfied by a system under all circumstances.

- **Priority Values take Precendence**: Conflicts beween multiple mandatory goals or multiple goal clusters are resolved based on prioritiy values assigned by the stake holders.

Incorporating more advanced and state-of-the-art conflict detection and resolution mechanisms is outside the scope of this thesis and can be viewed as a potential future research topic.

3.6 Summary

This chapter provided an overview of the proposed decision making model. We began this chapter by describing the components of a decision-making model. Next, we highlighted the salient features of GAAM model and outlined the advantages and drawbacks of this model, and presented certain enhancements that can be applied to it. We then discussed the requirements of a comprehensive and well-designed decision-making model. Subsequently we presented the architecture of our decision-making model and described how a decision-making process could use this model to acheive adaptation. The next chapter describes how the proposed decision-making model can be used to build a self-adaptive application.

Chapter 4

Developing Self-Adaptive Applications

Developing an application with self-managing capabilities, or converting a legacy system to an autonomic system is challenging and should be carried out according to a well-defined process. Various approaches and models have been proposed by researchers to address self-* properties in self-adaptive systems. This chapter describes how a self-managing application can be developed and configured to use the proposed decision-making model. We begin this chapter by describing StarMX a Java-based adaptation framework developed for self-adaptation systems. Next, we describe the process of integrating StarMX with the proposed decision-making model to build an adaptation manaer. Subsequently we explain the adaptation manager configuration properties and then describe the deployment options available for the adaptation manager. We conclude this chapter by describing the steps required to build a self-adaptive application using the developed adaptation manager.

4.1 StarMX - A Java-based Adaptation Framework

In[3], Asadollahi et al. have presented a framework called StarMX for developing self-managing Java-based systems. StarMX is a generic and configurable framework that facilitates the creation of self-adaptive software applications. It incorporates Java Management Extensions (JMX) technology and is capable of integrating with various policy and rule engines. Although originally designed for Java EE systems, the incorporation of JMX in Java SE (after J2SE 5.0), has enabled the framework to be used for all Java-based systems. The StarMX framework does not enforce any specific approach or algorithm, and it aims to provide sufficient flexibility for the developer to apply different composition patterns and adaptation mechanisms[3].

Figure 4.1 shows the high-level view of the StarMX architecture. The architecture consists of two main elements namely the *execution engine* and a set of *services*. The Execution Engine module realizes the adaptation processes. It executes the adaptation logic defined by the application developer to adapt the system based on its current situation utilizing services provided in the service layer.

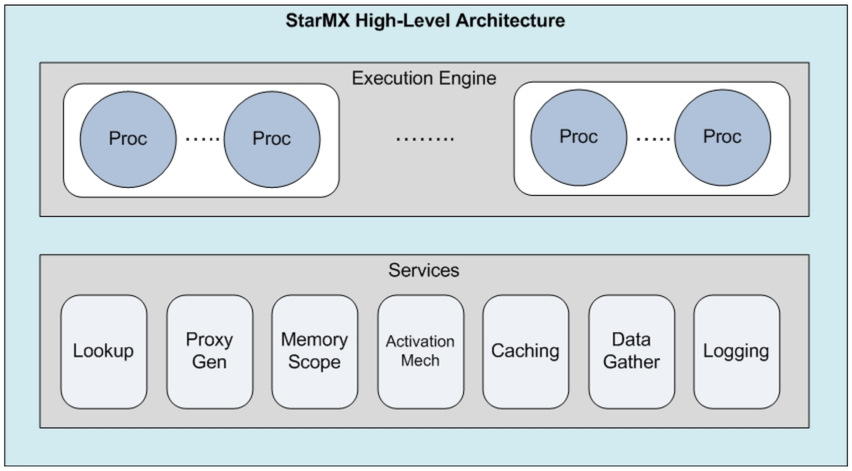


Figure 4.1: StarMX High-level Architecture

The two key components of the execution engine are *process* and *execution chain*. Figure 4.2 shows the architecture of this part of the framework and its interaction with the adaptable software. Processes comprise the building blocks of an adaptation manager. Each process may represent a single function or a group of consecutive modules of adaptation mechanism. The processes are chained together to form execution chains. The execution chains act as an adaptation manager and are associated with an individual activation mechanism. When activated, the processes in the chain execute sequentially. As shown in figure 4.2, each process needs a collection of objects, called *anchor objects*, to perform its task. Either service-providing helper objects or sensors/effectors of the underlying adaptable resource can function as anchor objects. A framework configuration file called *starmx.xml* facilitates the definition of the required set of anchor objects for each process and their lookup information. At execution time, the execution engine creates and inserts the anchor objects into the execution process.

In order to support standard forms of access to the anchor objects, StarMX offers the ability to define MBeans or JavaBeans. There are two approaches for defining a process namely using a policy language and using Java code by implementing an available interface. In the first case, the user defines the slef-adaptive system requirements in the form of action policies. StarMX employs the Adapter design pattern to interact with external policy

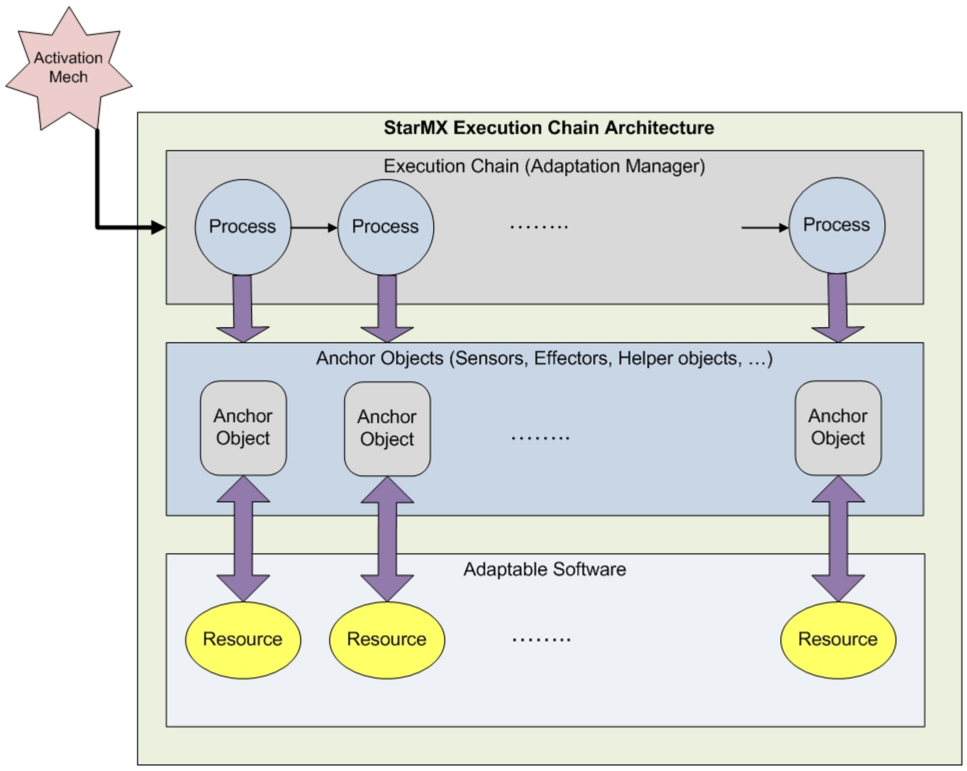


Figure 4.2: StarMX Execution Chain Architecture

engines. StarMX also allows for the implementation of a process using the Java language to allow ultimate flexibility in rule specification. The composition of processes in execution chains to build adaptation managers can be either static or dynamic. In static mode, the chain of processes is pre-configured, while in dynamic mode, several execution chains may form an adaptation manager on-the-fly. The execution chain architecture provides a high degree of flexibility in the design of adaptation manager. It allows the definition of all adaptation mechanism modules in a single process, or the design of an arbitrary number of chained processes for this purpose.

The run-time behavior of the framework consists of the following phases:

- **Start up**: At startup, StarMX prepares the environment for the optimized operation of the execution chains. All services are initialized, and processes and execution chains are deployed based on the specified properties.
- **Operation**: Once the framework has successfully started, it moves to the operation phase where the execution chains are activated. Upon activation, the processes in the

chain are executed in order, providing each process with the required set of anchor objects. The process invokes the anchor objects' methods to obtain data from or to send commands to the application. If a policy language is used, the relevant policy engine is invoked through its adapter for policy execution. If custom Java code has been written to invoke a custom adaptater, the custom Java code is invoked to initialize the custom adapter. The process can also use memory scopes to keep or share data with other processes.

- **Shutdown**: Finally, during the shutdown phase, the framework undeploys the execution chains and processes, and ceases to work.

The next section describes the mechanism used to interface the proposed decision-making model with StarMX.

4.2 Building an Adaptation Manager

An adaptation manager is composed of an adaptation framework and a decision-making model. The adaptation framework provides the ability to periodically monitor the adaptable software, provides access to the data produced by the sensors, and facilitates the execution of an effector action that attempts to restore the system to normal. The decision making model interprets the data produced by the sensors to determine if the system is operating within its designated limits. If the system needs adaptation, the adaptation manager utilizes the decision making model to determine the action that must be performed by the effectors to bring the system back to normal.

In the previous chapter we presented a generic, enhanced goal-oriented decision-making model. However, to use this model in an adaptation manager, we need an adaptation framework. In order to interface the proposed decision-making model with StarMX, periodically executing processes must be defined within StarMX. As mentioned in section 4.1, StarMX provides the ability to interface custom Java-based policy adapters, and this is the mechanism that we have employed to interface the proposed decision-making model with StarMX. We have implemented a Java-based adapter that functions as a bridge between StarMX and our decision-making model. It should be noted however, that the implementation of the decision-making model is generic enough to be used with any adaptation framework and is not dependant on the workings of StarMX.

The next section describes the configuration information required for the adaptation manager consisting of StarMX and the proposed decision-making model.

4.2.1 Configuration

In order to configure an adaptation manager that leverages StarMX as the underlying adaptation framework along with the proposed decision-making model, we need to provide configuration information for both components.

StarMX Configuration

StarMX contains information about the manageability endpoints of the adaptable software. MBeans representing the application instrumentation interfaces along with their connection information are defined in the StarMX framework configuration file *starmx.xml*. This is an XML-based configuration file and contains information about the StarMX processes, execution chains, MBeans etc. The MBeans contain accessor and mutator methods for obtaining and changing the exposed properties of the adaptable software.

In addition to the MBean information, a process using a custom Java-based policy adapter must be specified, along with the name of the file containing the configuration information for the proposed decision-making model. We have modified the StarMX source code such that setting the *policy-type* of the *process* tag to *"gaam"* will invoke the relevant custom Java-based policy adapter. Furthermore, an execution chain consisting of the aforementioned process should also be defined in the file. A snippet of a sample process and execution chain configuration has been provided below:

```
<process id="rule1" policy-type="gaam" policy-file="gaam.xml">
</process>
<execute>
    <timer-info interval="15" unit="second"/>
    <processref refid="rule1" />
</execute>
```

Decision-Making Model Configuration

The decision-making model (DMM) configuration information is defined in an XML-based configuration file whose name is specified in *starmx.xml*. This configuration file contains the language for defining the information of attributes, actions, goals and goal clusters via XML tags. A summary of the key configuration items is presented in tables 4.1, 4.2 4.3, 4.4 and 4.5. A complete example of this file is also available in Appendix A.

The next section describes the run-time behavior of the adaptation manager consisting of StarMX and the proposed decision-making model.

Table 4.1: DMM Configuration - Attributes and Actions

DMM Entity	**XML Entity**	**XML Attributes**
Attribute	*attribute*	• *attribute-id*: Attribute identifier • *mbean-id*: MBean identifier as defined in *starmx.xml* • *mbean-attribute-name*: MBean attribute name • *mbean-attribute-type*: MBean attribute data type. Valid values are *Integer,Double,Float,Boolean,String,Object*
Action	*action*	• *action-id*: Action identifier. • *action-type*: Contains the action type, since the same XML entity is used to represent invoker, atomic and composite actions. Valid values are *invoker,atomic,composite.* • *mbean-id*: MBean identifier as defined in *starmx.xml* • *mbean-attribute-name*: MBean attribute name • *mbean-attribute-type*: MBean attribute data type. Valid values are *Integer,Double,Float,Boolean,String,Object.* This property is only specified for atomic actions. • *mbean-method-name*: MBean method name. This property is only specified for invoker actions. • *mbean-method-signature*: MBean method signature. This property is only specified for invoker actions. • *sub-action-id*: Comma delimited list of sub-action identifiers for composite actions.

Table 4.2: DMM Configuration - Goal Attributes and Goal Actions

DMM Entity	**XML Entity**	**XML Attributes**
Goal Attribute	*goal-attribute*	• *attribute-id*: Attribute identifier • *threshold-min*: Minimum threshold value for attribute. • *threshold-max*: Maximum threshold value for attribute. • *threshold-val*: Exact threshold value for attribute. • *threshold-val-isnull*: The goal attribute is active if the attribute value is null. Valid values are *true, false* • *custom-threshold-evaluator*: Contains the name of the custom Java-class used to evaluate the threshold value. • *realiant-attrib-ids*: Contains the identifiers of the attributes which should be passed as a parameter to the custom Java-based threshold evaluator class.
Goal Action	*goal-action*	• *action-id*: Action identifier. • *preference*: Contains the preference value.

Table 4.3: DMM Configuration - Goal Action Properties and Preconditions

DMM Entity	**XML Entity**	**XML Attributes**
Goal Action Property	*goal-action-property*	• *mbean-attribute-value*: Contains the value to be passed as a parameter to the effector method. • *mbean-attribute-value-isnull*: Indicates that a null value must be set by the effector action. Valid values are *true, false* • *custom-mbean-attribute-value-builder*: Contains the name of the custom Java-class used to build the value to be passed as a parameter to the effector method.
Goal Action Precondition	*goal-action-precondition*	• *reliant-attrib-ids*: Contains a comma delimited list of the attribute identifiers that the precondition depends upon. • *reliant-attrib-vals*: Contains a comma delimited list of the values of the attributes that the precondition depends upon. • *reliant-attrib-val-relation*: Contains the relationship that must exist between the attributes that the precondition depends upon. Valid values are *eq,neq,gt,gte,lt,lte* for equals, not equals, greater than, greater than or equals, less than, less than or equals. • *custom-precondition-evaluator*: Contains the name of the custom Java-class used to evaluate the precondition for this goal action.

Table 4.4: DMM Configuration - Mandatory Goals, Negotiable Goal Clusters and Goals

DMM Entity	**XML Entity**	**XML Attributes**
Mandatory Goal	*mandatory-goal*	• *goal-id*: Contains the goal identifier. • *priority*: Contains the goal priority used for conflict detection and resolution. • *activation-function-operation-type* Contains the activation function operation type. Valid values are *and, or*.
Negotiable Goal Cluster	*negotiable-goal-cluster*	• *cluster-id*: Contains the cluster identifier. • *priority*: Contains the clusterl priority used for conflict detection and resolution. • *voter-type* Contains the type of voting algorithm. Valid values are *borda-count, winner-takes-all,custom*. • *custom-voter*: Contains the name of the Java-class containing the customizable voting algorithm.
Negotiable Goal	*negotiable-goal*	• *goal-id*: Contains the goal identifier. • *goal-priority*: Contains the goal priority used by the action selection process. • *activation-function-operation-type* Contains the activation function operation type. Valid values are *and, or*.

Table 4.5: DMM Configuration - Adaptive Negotiable Goal Clusters and Goals

DMM Entity	**XML Entity**	**XML Attributes**
Adaptive Negotiable Goal Cluster	*adaptive-negotiable-goal-cluster*	• *cluster-id*: Contains the cluster identifier. • *priority*: Contains the clusterl priority used for conflict detection and resolution. • *voter-type* Contains the type of voting algorithm. Valid values are *borda-count*, *winner-takes-all*,*custom*. • *custom-voter*: Contains the name of the Java-class containing the customizable voting algorithm. • *rl-agent-type* Contains the type of reinforcement learning algorithm. Valid values are *q-learning*, *sarsa*,*custom*. • *custom-rl-agent*: Contains the name of the Java-class containing the customizable reinforcement learning algorithm.
Adaptive Negotiable Goal	*adaptive-negotiable-goal*	• *goal-id*: Contains the goal identifier. • *goal-priority*: Contains the goal priority used by the action selection process. • *activation-function-operation-type* Contains the activation function operation type. Valid values are *and*, *or*.

4.2.2 Run-time Behavior

The run-time behavior of the developed adaptation manager consisting of StarMX and the proposed decision-making model is composed of the following phases:

- **Start up**: When StarMX initializes, it initiliazes the custom Java-based policy adapter among other components. Upon initialization of the custom policy adapter, the adapter loads and parses an XML based configuration file (see section 4.2.1 for details) containing information about the constituent components of the decision-making model to build objects representing these components i.e. the attributes, actions, mandatory goals, goal clusters etc. and stores the information about these objects in its execution context.
- **Operation**: Once the system has been initialized, the StarMX processes begin their periodic execution. During each process execution phase the custom adapter obtains the list of all attributes defined in the system and invokes the sensor methods associated with the attributes to obtain the value of each attribute. The values of the attributes are then pushed to the decision-making model. The decision-making model analyzes the attributes and determines the appropriate effector actions to be invoked using the mechanism outlined in section 3.5. The effector actions are then executed via StarMX in an attempt to restore normalcy to the system. This cycle is repeated for each periodic process execution phase.
- **Shutdown**: During shutdown, StarMX invokes shutdown operations on the custom Java-based policy adapter among other components.

The next section describes the deployment options available for the adaptation manager.

4.2.3 Deployment Options

An adaptation manager consisting of StarMX as the adaptation framework and the proposed decision-making model can support two deployment options: *local* and *remote*. In local deployment, that adaptation manager is deployed on the same server and the same JVM that the adaptable software executes on. In remote deployment, the adaptation manager is deployed on a different server and started as a separate application that manages the target system. Selecting the appropriate deployment option in a real environment is a trade-off among the performance overhead of each approach, ease of deployment, and other domain-specific concerns[3].

For example, in local mode, the adaptation manager consumes resources (e.g. CPU and memory) on the system where the adptable software is deployed. In remote mode, only the sensing and effecting operations use the system resources accessed by the adaptable software. On the other hand, in remote mode the speed of the adaptation process may be less than the local node if network latency comes into play, which may render remote deployment as a non-option for mission critical applications.

The next section outlines the steps required to build a self-adaptive application leveraging the aforementioned adaptation manager.

4.3 Building Self-Adaptive Applications

Creating self-managing software systems, regardless of the techniques or models that are used for dynamic problem detection and resolution, requires a systematic approach that helps the developer to proceed incrementally to achieve the final result. In an abstract comparison, it is similar to a software development methodology or process, which starts from the requirement specification and ends at deployment. Based on our experience in enabling systems with adaptation behavior, we defined a six-step process to build self-managing systems using the StarMX framework and the proposed decision making model. Figure 4.3 demonstrates these steps and their input and output artifacts.

- **Step 1: Specifying self-managing requirements** - Self-managing solutions mostly focus on non-functional and QoS (Quality of Service) requirements. Therefore, these solutions deal with performance, security, reliability or other quality factors. These requirements are also referred to as self-* properties. The goal of this step is to clearly specify or model these requirements and the expected objectives for the target system.

  - *Inputs*: Service Level Agreement (SLA), which is decomposed into Service Level Objectives (SLO). Other non-functional requirements can also be considered as the input to this phase.
  - *Outputs*: Self-managing requirement specification.

  The way that these requirements should be specified or modeled is an open research area and is not within the scope of this work.

- **Step 2: Transforming requirements into decision-making model components** - After the self-adaptation requirements are specified, they need to be transformed such that they can be represented by the proposed decision making model. The goal of this step is to convert the requirements into goals and then determine the attributes and actions for each goal. In order to determine the actions and attributes, a thorough feasiblity analysis must be done on the existing application (in the case

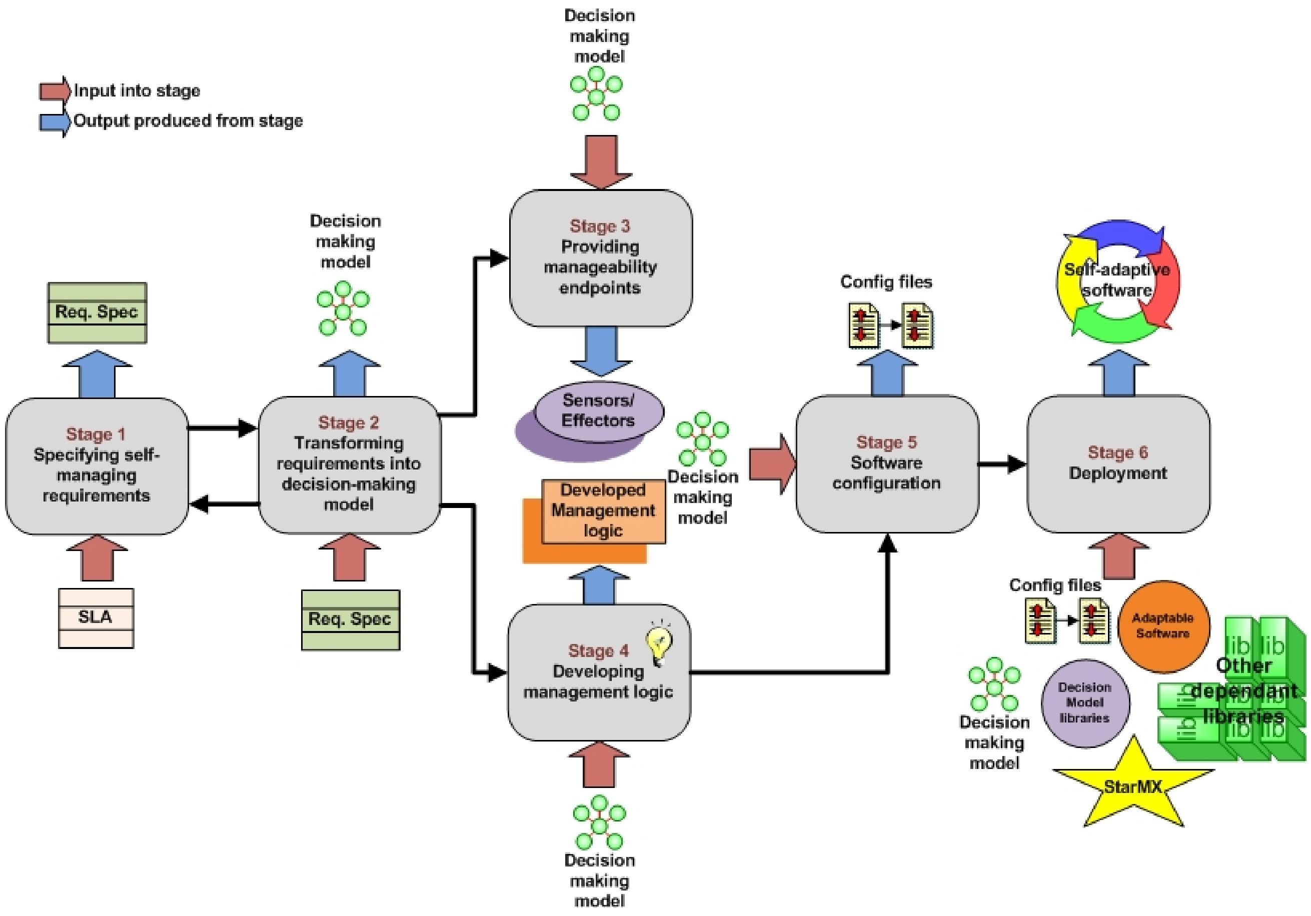


Figure 4.3: Steps to Develop Self-Adaptive Software

of legacy applications) or functional and design specifications of the application (in the case of new applications) to determine if the required attributes and/or actions can be obtained or performed by the system.

- *Inputs*: Self-managing requirement specification.
- *Outputs*: Decision making model core components i.e. goal clusters, goals, attributes and actions.

Based on feedback from this step, the self-managing requirements may need to be modfied. This may happen if the feasibility analysis determines that certain attributes or actions required by the goals cannot be obtained or performed. Since a goal represents a self managing requirement, if its dependencies cannot be satisfied, it implies that the requirement may not be achievable.

- **Step 3: Providing manageability endpoints** - Manageability endpoints (sensors and effectors) are the gateways for interfacing with a resource for management purposes. Based on the attributes and actions required by all the goals in the decision-making model, the required set of sensors and effectors are identified. These objects should be designed and instrumented into the target application in this phase.
  - *Inputs*: Attributes and actions for all goals in the decision-making model.
  - *Outputs*: Sensors and effectors.

  As mentioned earlier, we use StarMX as the underlying adaptation framework and it will be used for providing access to the sensors and effectors. Since StarMX supports both JMX MBeans or simple Java objects, the sensors and affectors can be developed by either methodology.

- **Step 4: Developing management logic** - This is the core part of the process for building a self-managing system. The management logic needs to be specified from the perspective of StarMX in addition to the proposed decision-making model. The StarMX management logic is developed as a set of Java-based processes, policies, and anchor objects that form control loops. The management logic for the proposed decision-making model is developed by identifying the attribute thresholds, action preferences, goal and goal cluster priorities, voting and reinforcement learning algorithms etc., and developing any required Java-based classes for the management logic pertaining to the core components (e.g. custom classes for determining if the threshold of attributes have been exceeded, etc.).
  - *Inputs*: Decision making model core components and sensors and effectors.
  - *Outputs*: Developed management logic (for StarMX and the proposed decision-making model).

- **Step 5: Configuring the software** - The next step is to define configuration files for StarMX and our decison-making model. StarMX uses an XML configuration file containing information of the anchor objects, processes, and execution chains, MBeans and MBean servers called *starmx.xml*. The information about the goal clusters, goals, actions and attributes can be specified in a configuration file whose name must be provided in the *starmx.xml* file.
  - *Inputs*: Developed management logic (for StarMX and the proposed decision-making model).
  - *Outputs*: Valid configuration files (for StarMX and the proposed decision-making model).

- **Step 6: Deployment** - The last step is to integrate the framework with the target system, deploy it, and test whether the created self-managing system performs as expected.
  - *Inputs*: Target system and dependant libraries, framework; libraries for StarMX and the proposed decision-making model; valid configuration files (for StarMX and the proposed decision-making model), etc.
  - *Outputs*: The self-adaptive system.

4.4 Summary

In this chapter we have outlined the entire process of building self-adaptive software using the proposed decision-making model. We began the chapter by describing StarMX - an adaptation framework for Java-based self-adaptive systems. We then described the process of building an adaptation manager consisting of StarMX and the proposed decision-making model. We concluded the capter by defining a systematic step-by-step approach for building self-adaptive software. The next chapter describes the experimental studies that have been performed by us to evaluate the effectiveness of the decision making model developed in this dissertation.

Chapter 5

Experimental Studies

This chapter aims to provide an evaluation of the decision-making model developed in this thesis. In this experimental evaluation, the following research questions are taken into account:

- **RQ1**: What is the impact of the adaptation technique on the systems goals? Does it improve the goal satisfaction level comparing with the non-adaptive case?
- **RQ2**: What is the impact of making control loops a first class entity in a decision-making model? Does a decision-making model based on mandatory and adaptive negotiable goals outperform a decision-making model based execlusively on mandatory goals or negotiable goals?

A news web application and an IP telephony system are used as case studies to evaluate and analyze the decision-making model through a set of experiments that focus on the research questions. We begin this chapter by presenting a brief description of the two case studies. We then describe the steps involved in conversion of the applications into self-adaptive applications. Subsequently we describe the experiments conducted on the self-adaptive applications with different types of decision-making model consisting exclusively of mandatory goals, negotiable goals or a combination of mandatory, negotiable and adaptive negotiable goals. Lastly we present the results of our experiments and the conclusions deduced from the experiment results.

5.1 Case Study 1: News Web Application

Consider a legacy news web application which provides a number of services like viewing news, weather conditions, stock information and searching. The web application contains components with multiple implementations, each optimized for a particular workload.

Components of a higher quality require more resources than their lower quality counterparts. However the task of switching between these components based on the workload is currently a manual one. The challenge at hand is to make this system adaptable so that it can effectively process varying workloads in an automated fashion.

5.1.1 Motivation

The motivation for choosing a news web application was the problems encountered by news web sites in the US after 9/11. The news web sites usage skyrocketed on that day, and continued throughout the week. A Los Angeles Times report offers a few numbers: "MSNBC saw a tenfold increase in traffic, with as many as 400,000 hits at any point. CNN.com surged to 162.4 million page views in 24 hours from a 14 million average." In order to deal with this situation, the technical staff redesigned the site in an effort to remove all of the extraneous information, and concentrating on the bare facts. The CNN web site, which normally includes pages with various links and graphics, was reduced to one breaking news web page. While the changes were made to the application by the administrators, we are interested in adding an adaptation manager to accomplish this task. This scenario is appropriate to show how effective the adaptation mechanism would be for satisfying the goals of end-users and administrators.

In fact, during the 9/11 event, administrators and network managers tried to manage the system by tuning the parameters and by applying server and network-level actions. But application-level actions were missing, which were applied manually in the CNN case. Therefore, the interesting question for us is how adaptation actions at the application level would impact the system behavior. Although, other actions are applicable to this case, we focus only on the application-level adaptation actions.

5.1.2 Original Application Architecture

The news web application was developed by us and is Java-based. It is deployed on the JBoss web application server. It uses MySQL as its backend database. Figure 5.1 shows the original high level architecture of the system. The architecture is divided into three tiers namely the presentation tier which displays the information to the user, the business logic tier which obtains the information from and writes the information to the database tier, and the database tier which stores the information. The information pertaining to news, stocks and weather is stored in the database. The News Items EJB, Stocks EJB and Weather EJB obtain information related to news, stocks and weather from the database. The EJBs use the Java Persistence API (JPA) to communicate with the underlying database. The web pages are implemented using JSPs.

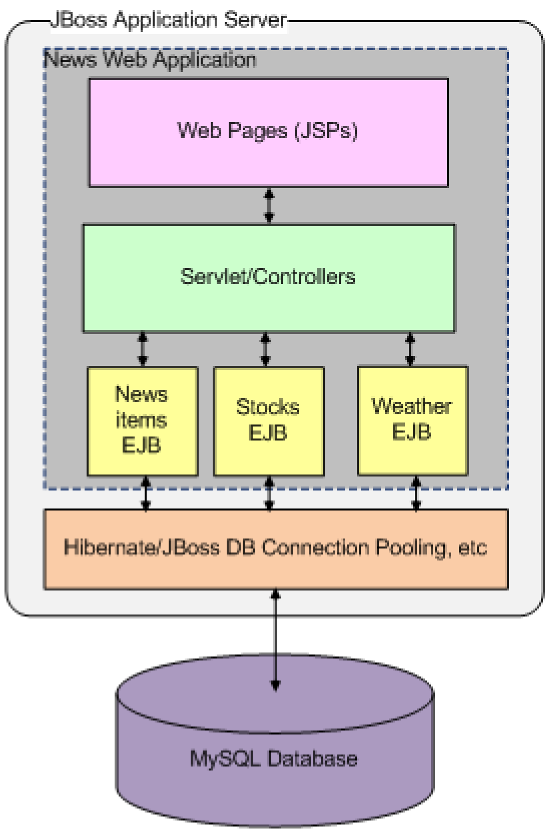


Figure 5.1: News Web Application System Architecture

At a high level, the workings of the system are summarized as follows:

1. When a user accesses the news website, it causes the client web pages to send a request to the servlet.
2. The servlet processes this request and communicates with the EJBs to obtain information.
3. The EJBs obtain the information from the database and return the information to the servlet.
4. The servlet returns the information to be displayed on the web pages and the user can read the contents of the news website.

The next section describes how the news web application can be converted into a self-adaptive application.

5.1.3 Self-Adaptive Application Architecture

The challenge at hand is to make this system adaptable so that it can effectively process varying workloads in an automated fashion. We use the 6 development steps covered in section 4.3 to make this application self-adaptive.

- **Step 1: Specifying self-managing requirements** - In this step we convert the SLA into adaptation requirements. As per the SLA, when the system is under a normal workload, it is expected to display high quality components, and when subjected to a higher workload it can begin a gradual degradation of the offered services and can eventually degrade to a text-only website. However, at no point in time should the system render itself completely non-responsive except if the application server on which the application is deployed crashes. Hence, the self-adaptive news web application has three adaptation requirements namely:
  - **High Service Availability**: this is a mandatory requirement.
  - **Maximum Throughput**: this is a negotiable requirement.
  - **Reasonable Response Time**: this is a negotiable requirement.
- **Step 2: Transforming requirements into decision-making model components** - Based on the specified adaptation requirements, we need to identify the system goals, goal clusters, attributes and actions. Since the system has a mandatory requirement of high service availability, this requirement is transformed into a mandatory goal indicating that the system must always be up and running, and if the current load exceeds a maximum threshold, then all the offered services must be suspended to prevent the system from crashing. Additionally, since the system has related negotiable requirements of maximum throughput and a reasonable response time, these requirements can be transformed into a negotiable or adaptive negotiable goal cluster. The comprising goals of the goal cluster would indicate the quality of components to be displayed by the EJBs based on the current load and current response time of the system. Hence the goals and goal clusters of the decision-making model can be represented as follows:
  - **Mandatory Goal** MG_1: System must always be up and running; so if the load exceeds a maximum value (e.g. 100 concurrent connections), stop all services.
  - **(Adaptive) Negotiable Goal Cluster** GC_1:
    - **(Adaptive)Negotiable Goal** G_1: If the current load is low, throughput is high, and response time is less than e.g. 2 seconds, display best quality components.

    * **(Adaptive)Negotiable Goal** G_2: If the current load and throughput are medium and response time is less than e.g. 3 seconds, display medium quality components.
    * **(Adaptive)Negotiable Goal** G_3: If the current load is high, throughput is low and response time is less than e.g. 4 seconds, display low quality components.
    * **(Adaptive)Negotiable Goal** G_4: If the current load is high, throughput is low, and response time is greater than e.g. 4 seconds, display a text-only page.

  In order to achieve these goals, we need the ability to determine the overall system response time i.e. the time taken to service a client request, the ability to determine the system throughput i.e. number of bytes returned per second and the ability to determine the type of components being displayed by the system. These form the attributes of the system that need to be monitored. Hence the attributes of the decision-making model can be represented as follows:

  - **Attribute** At_1: Obtain the current load.
  - **Attribute** At_2: Obtain the current response time.
  - **Attribute** At_3: Obtain the current NewsEJB type.
  - **Attribute** At_4: Obtain the current WeatherEJB type.
  - **Attribute** At_5: Obtain the current StocksEJB type.

  In order to achieve the goals, we also need the ability to modify the type of components being displayed by the system since this affects the response time, throughput and availability of the system. Additionally, we also need the ability to stop all services being offered by the system. These form the system actions that can be performed to attain the aforementioned system goals. All of these actions are atomic actions. Hence the atomic actions of the decision-making model can be represented as follows:

  - **Action** Ac_1: Modify the current NewsEJB type.
  - **Action** Ac_2: Modify the current WeatherEJB type.
  - **Action** Ac_3: Modify the current StocksEJB type.
  - **Action** Ac_4: Stop all system services.

- **Step 3: Providing manageability endpoints** - After identifying the goals, actions and attributes, we need to instrument the required set of management interfaces (sensors and effectors). As discussed earlier, the services in the web application are provided by four components. Hence, we need a set of sensors and effectors to control the EJBs and the servlet. Since Java allows management of any component through MBeans, four MBeans (one for each component) are designed to provide information about the average response time, throughput and component quality as described below:

- **ServletMBean**: Manages the servlet and provides the following capabilities:
  - Contains a *getter* method for obtaining the system response time.
  - Contains a *getter* method for obtaining the system throughput.
  - Contains a *setter* method for stopping all services offered by the system.
- **NewsMBean**: Manages the NewsEJB and provides the following capabilities:
  - Contains a *getter* method for obtaining the component type.
  - Contains a *setter* method for modifying the component type.
- **WeatherMBean**: Manages the WeatherEJB and provides the following capabilities:
  - Contains a *getter* method for obtaining the component type.
  - Contains a *setter* method for modifying the component type.
- **StocksMBean**: Manages the StocksEJB and provides the following capabilities:
  - Contains a *getter* method for obtaining the component type.
  - Contains a *setter* method for modifying the component type.

- **Step 4: Developing management logic** - As mentioned before, this is the core part of the process for building a self-managing system. We specify the management logic from the perspective of StarMX in addition to the proposed decision-making model. The aforementioned MBeans are registered as anchor objects within an instance of the StarMX. Processes are defined and configured to use the ABLE policy engine or the decision-making model developed in this thesis to realize the adaptation process. Execution chains are defined to periodically invoke the processes and thus execute the complete Monitor, Anakyze, Plan and Execute (MAPE) loop. We use ABLE in order to facilitate a comparative study of the proposed decision-making model against a well-established adaptation engine.

- **Step 5: Configuring the software** - In this step, XML-based descriptions of the elements defined in the above two steps were added to the configuration files.
  - An example XML configuration for attribute At_1 is as follows:

```
<attribute attrib-id="at_1"
    mbean-id="MainServletMBean"
    mbean-attribute-name="CurrentLoad"
    mbean-attribute-type="Integer" />
```

- An example XML configuration for action Ac_4 is as follows:

```
<action action-id="ac_4"
    action-type="atomic"
    mbean-id="MainServletMBean"
    mbean-attribute-name="StopService"
    mbean-attribute-type="Boolean" />
```

- An example XML configuration for mandatory goal MG_1 is as follows:

```
<mandatory-goal goal-id="MG_1"
    <goal-attribute attribute-id="at_1" threshold-max="100" />
    <goal-action action-id="ac_4">
        <goal-action-prop mbean-attribute-value="true" />
    </goal-action>
</mandatory-goal>
```

Appendix A contains a complete sample XML configuration file for the news web application.

- **Step 6: Deployment** - In this case study, the adaptation manager consisting of StarMX and ABLE/the proposed decision-making model is setup as an independant executable to allow the flexibility to run the external adaptation engine on a separate machine if required. This enabled evaluation of the remote deployment option.

Figure 5.2 shows the modified self-adaptive architecture of the system. At a high level, the workings of the self-adaptive system are summarized as follows:

1. Periodically, information about the application's run-time dynamics are obtained by the StarMX application. This information is analyzed by the configured adaptation process (the ABLE policy engine or the decision making model developed in this thesisis) and effectors are invoked to select the components that must be displayed by the web application, when it receives a user request.

2. When a user accesses the news website, it causes the client web pages to send a request to the servlet.

3. The servlet processes this request and communicates with the EJBs to obtain information.

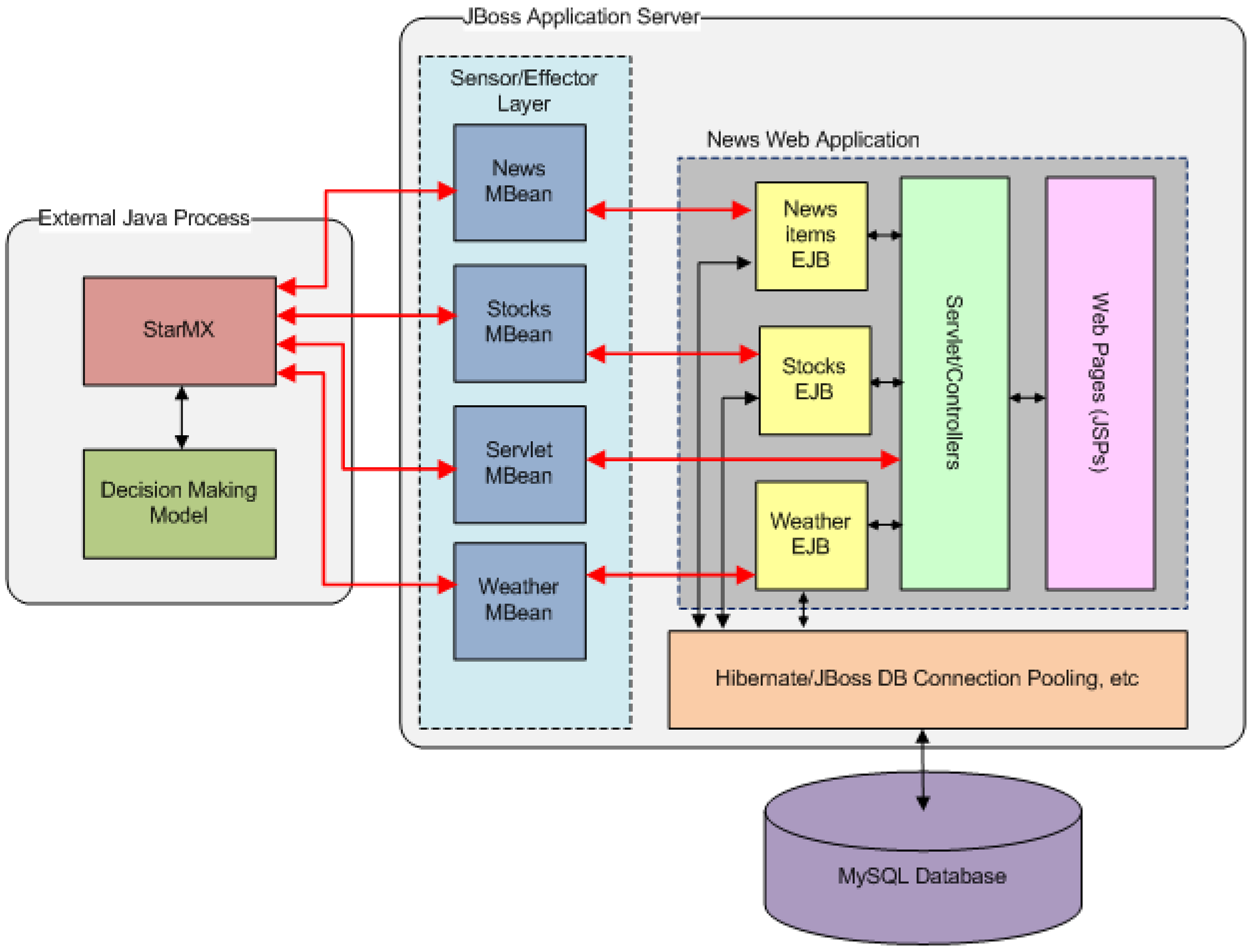


Figure 5.2: Self-Adaptive News Web Application Architecture

4. The EJBs invoke methods on their respective MBeans to determine the version of the component to be displayed, obtain the selected component information from the database and return the information to the servlet.

5. The servlet returns the information to be displayed on the web pages and the user can read the contents of the news website.

The next section describes the experiments conducted on this case study to evaluate the effectiveness of the proposed decision-making model.

5.1.4 Experiment Design

In order to evaluate the effectiveness of the decision-making process utilized by the self-adaptive news web application, we need to be able to examine the application behavior under varied workloads. A large number of commercial software products are available for stress testing web applications. However most of these products required expensive software licenses and hence could not be used our experimental evaluation. We developed a simple Java-based application called *JLoadRunner* that simulates user interaction with a web-site. *JLoadRunner* can be configured to create a large number of web-crawler threads. Each thread can be configured to run for a specific number of iterations. In each iteration the web crawler thread connects to the news web sites, accesses several pages and images and calculates the response time and throughput of the web site.

We deployed the self-adaptive news application and *JLoadRunner* on the same machine. By tuning the configuration parameters of *JLoadRunner*, the system was subjected to stress tests involving varying degrees of concurrent loads. We measured the system response time and throughput for each stress test iteration. We also changed the adaptation technique used by the application for each stress test iteration.

The application was first tested with no adapation to get the baseline numbers. Next the decision making model was configured to use an adaptation technique employing only mandatory goals specified as action policies processed by the ABLE policy engine. Subsequently the decision making model was configured to use an adaptation technique employing mandatory goals and negotiable goal clusters. Lastly, the decision making model was configured to use an adaptation technique based on mandatory goals and adaptive negotiable goal clusters. Two reinforcement learning algorithms namely Q-Learning and SARSA were used for the adaptive negotiable goal clusters. We used two different algorithms to examine the impact of using off-policy and on-policy algorithms respectively.

To minimize the experimental errors due to sporadic events, three replications were conducted for each stress test iteration. The specification of the machine used to conduct the experiments was: Windows Vista Home Premium, Intel Core 2 Quad CPU T550 @ 1.66GHz, 2GB of RAM.

5.1.5 Obtained Results

Figures 5.3 and 5.4 display the obtained response times and throughput of the news web application when subjected to increasing amounts of concurrent workloads. The obtained results clearly demonstrate the expected i.e. when the workload experienced by the news web application increases, the response time increases and the throughput decreases. However, the response time of a news web application that is not self-adaptive increases by a significant amount as compared to the self-adaptive counterparts.

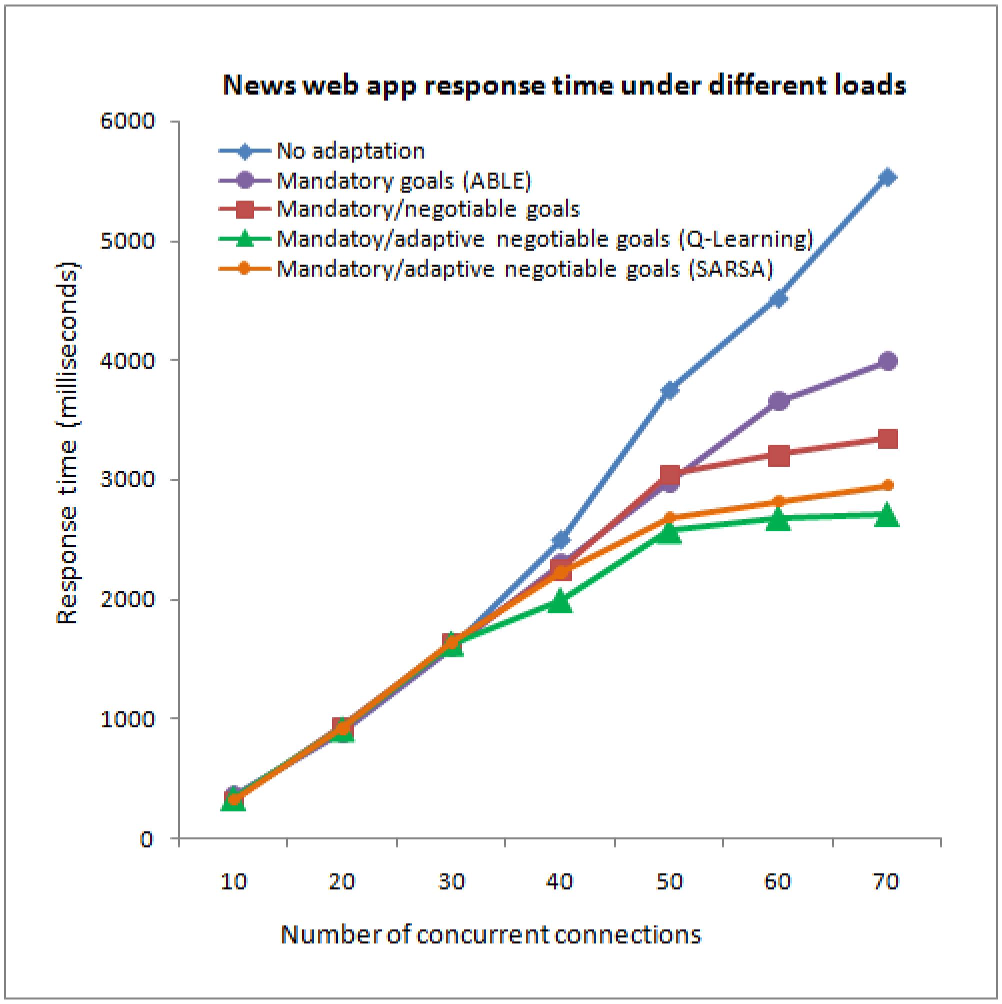


Figure 5.3: News Web Application: Response Time

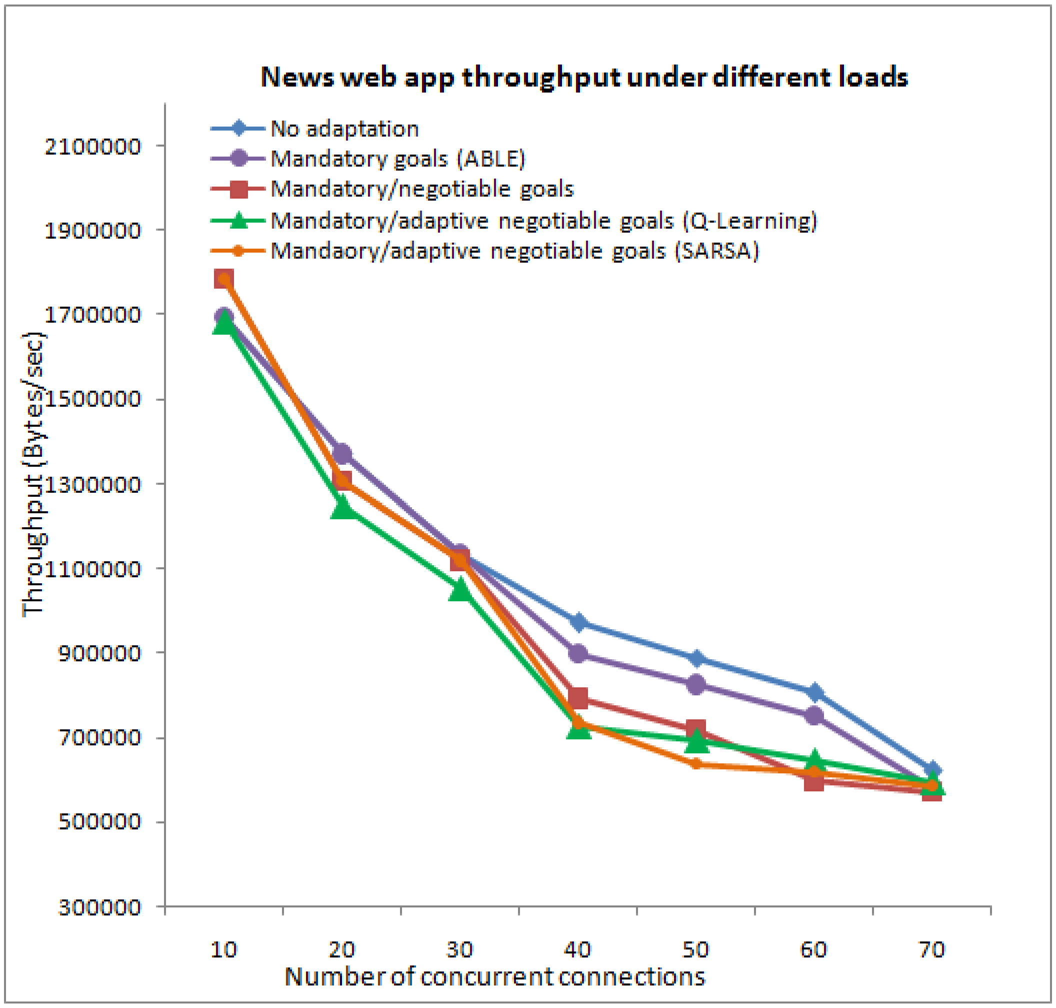


Figure 5.4: News Web Application: Throughput

Moreover the throughput of a news web application that is not self-adaptive is initially better as compared to the self-adaptive counterparts. This is because the news web application that is not self-adaptive always displays web content of the highest quality. This implies that despite the fact that the non self-adaptive news web app may process a smaller number of worloads/connections as compared to its self-adaptive counterparts, each connection receives more data as compared to its self-adaptive counterparts, since the self-adaptive counterparts vary the quality of the news content based on the system load. However as the workload increases, the throughput of the web application that is not self-adaptive decreases at a much sharper rate as compared to its self-adaptive counterparts. The throughput of the self-adaptive versions stabilizes at a constant value since at very high workloads these versions display only text content to the users, which as we know is acceptable as per the non-functional specifications/adaptation requirements of the system.

Additionally, the results demonstrate that the overall response time of a decision-making model configured to use an adaptation technique based on mandatory goals and adaptive negotiable goals is better than that of a decision making model configured to use an adaptation technique employing mandatory and negotiable goals which is better than that of a decision-making model configured to use an adaptation technique employing only mandatory goals specified as action policies processed by the ABLE policy engine. The response times obtained for a decision-making model configured to use an adaptation engine based on mandatory goals and adaptive negotiable goals using Q-Learning and SARSA are comparable with Q-Learning only slightly outperforming SARSA.

Furthermore, the results demonstrate that the overall throughput of a decision-making model configured to use an adaptation technique employing only mandatory goals specified as action policies processed by the ABLE policy engine is better than must similar to the overall throughput of the non-self-adaptive version of the news web application, where the throughput decreases by a sharp amount when the system is subjected to higher workloads. Based on the results we can also infer that the overall throughput of a decision making model configured to use an adaptation technique based on mandatory goals and adaptive negotiable goals is very similar to that of a decision making model configured to use an adaptation technique employing mandatory and negotiable goals.

The next section describes our second case study used to evaluate the proposed decsion making model.

5.2 Case Study 2: IP Telephony System

Call Controller 2 (CC2) is a over IP prototype system. It is deployed on the Mobicents media server and designed based on a service oriented architecture. It basically provides four main services:

- **Regular VOIP Calls**: This is the most basic service provided by all VoIP software. A caller can call a callee to establish a conversation.
- **Call Forwarding**: If a callee is unavailable, CC2 will try to forward the call to the callee's backup address, if it has one.
- **VoiceMail**: A caller can leave a voice message if the callee is unavailable and has no backup address, but his/her voicemail is enabled.
- **Call Blocking**: If a caller is in the callee's blacklist, the call will be blocked.

Mobicents is the first and only open source VOIP Platform certified for JSLEE 1.0. JSLEE (JAIN Service Logic Execution Environment) is the Java implementation of SLEE. In the telecommunications industry, a SLEE is a high throughput, low latency event processing application environment. The JAIN SLEE specification 3 allows popular protocol stacks such as SIP 4 to be plugged in as resource adapters. The extensible standard architecture naturally accommodates integration points with enterprise applications such as Web, CRM or SOA end points.

Mobicents is deployed on JBoss and brings to telecom applications a robust component model and execution environment. It complements J2EE to enable convergence of voice, video, and data in next generation intelligent applications. One of the main components of JSLEE are Service Building Blocks (SBB), which are comparable to Enterprise Java Beans (EJB) in J2EE systems. Mobicents enables the composition of different SBBs such as call control, billing, user provisioning, administration, and presence sensitive features. Monitoring and management of Mobicents components comes out of the box via the SLEE standard, which is based on JMX and SNMP interfaces. In our experiments, we utilize its JMX-based management facility to manage the CC2 system dynamically.

5.2.1 Motivation

The choice of CC2 as our case study is justified by the following characteristics of CC2: It is an open source Java system, which allows us to investigate and modify its source code; It addresses a real business need (VoIP), rather than a hypothetical one; and It is a large-scale system (with 171K lines of code), which utilizes more features of our proposed decision making model for adaptation.

5.2.2 Original Application Architecture

Figure 5.5 shows the architecture of the CC2 application. The application consists of three key SBB components to address its main functionalities:

- **ForwardingSBB** which provides regular VoIP call and call forwarding services
- **VoiceMailSBB** which is responsible for the voice mail service
- **BlockingSBB** that enables the call blocking service

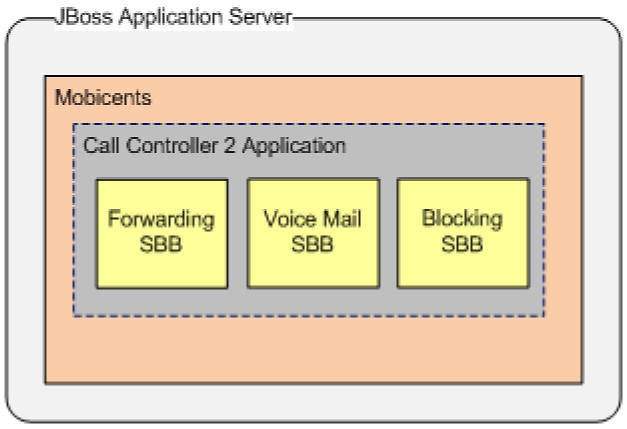


Figure 5.5: CC2 Application System Architecture

In order to make the adaptation scenarios more realistic, we modified CC2 to have users with different privileges, while preserving all other functionalities of the system. In the modified version, users are categorized into three classes: *Gold*, *Silver*, and *Bronze*, from highest to lowest priority. The gold users are the most valuable users of the VoIP system because they produce the most profit for the company. The Bronze users are the least profitable users, and silver users fall between the other categories. The system owner has to guarantee the quality of services provided to different classes of users, according to their contracts; otherwise, the owner is required to pay them a penalty. Moreover, all users are allowed to access all services provided by the system at all times.

5.2.3 Self-Adaptive Application Architecture

The challenge at hand is to make this system adaptable so that it can effectively process varying workloads in an automated fashion. We use the 6 development steps covered in section 4.3 to make this application self-adaptive.

- **Step 1: Specifying self-managing requirements** - The high level business objective of the new adaptable CC2 is to maximize the company's profit at different workload situations. To achieve this objective, the requirement is to maintain service availability such that it always results in the maximum benefit for the company. For this purpose, the system may decide to block access to a service for low-priority users at certain times (e.g. very high loads), in order to guarantee service quality for high-priority users. In other words, the service availability and self-optimizing property are the target objectives.

- **Step 2: Transforming requirements into decision-making model components** - Based on the specified adaptation requirements, we need to identify the system goals, attributes and actions. In this case there are two system goals namely high throughput and reasonable response time. In order to achieve these goals, we need the ability to determine the overall system response time for each call type i.e. the time taken to service a regular call, voice mail and call forwarding. We also need the ability to determine the system throughput i.e. number of successful call forward calls and regular calls. Additionally, we need to determine the category of the user placing the call in order to provide prioritized services. These form the attributes of the system that need to be monitored. In order to achieve the goals, we also need the ability to modify the voice mail level, call forward level and call blocking level. These form the system actions that can be performed to attain the aforementioned system goals.

- **Step 3: Providing manageability endpoints** - After identifying the goals, actions and attributes, we need to identify and instrument the required set of management interfaces (sensors and effectors). As discussed earlier, the services in CC2 are provided by three SBB components. Hence, we need a set of sensors and effectors to control these SBBs. The Mobicents server allows managing SBBs through MBeans. Three MBeans (one for each SBB) were designed to provide the average response time and throughput of each service and to block or unblock the service for each class of users. Note that each MBean acts as both sensor and effector.

- **Step 4: Developing management logic** - As mentioned before, this is the core part of the process for building a self-managing system. We specify the management logic from the perspective of StarMX in addition to the proposed decision-making model. The aforementioned MBeans are registered as anchor objects within an instance of the StarMX. Processes are defined and configured to use the ABLE policy engine or the decision-making model developed in this thesis to realize the adaptation process. Execution chains are defined to periodically invoke the processes and thus execute the complete MAPE loop. We use ABLE in order to facilitate a comparative study of the proposed decision-making model against a well-established adaptation engine.

- **Step 5: Configuring the software** - In this step, XML-based descriptions of the elements defined in the above two steps were added to the configuration files.
- **Step 6: Deployment** - In this case study, the adaptation manager consisting of StarMX and ABLE/the proposed decision-making model was deployed as a part of the CC2 application as opposed to being setup as an independant executable. This enabled evaluation of the local deployment option.

Figure 5.6 shows the modified self-adaptive architecture of the system.

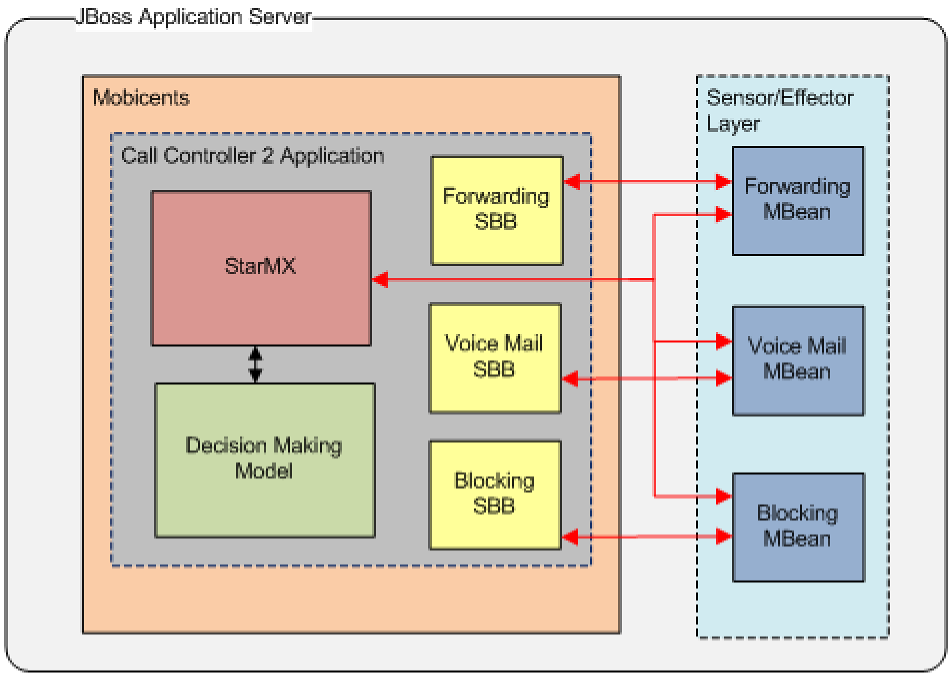


Figure 5.6: Self-Adaptive CC2 Application Architecture

The next section describes the experiments that were performed on this case study.

5.2.4 Experiment Design

In order to determine the impact of our adaptation technique on the system goals and to determine its overall effectiveness, we designed and implemented the following experiment:

We subjected the self-adaptive CC2 application to a series of stress tests with two different workloads namely a low load and a high load. The workloads were generated using SIPp 3.1, a free open source load generator for the SIP protocol. The workloads were defined as follows:

- **Low load**: this workload produces a specified number of requests on behalf of different classes of users (gold, silver, bronze) with a pre-defined time interval between requests. It is designed to be less than the capacity of the system for properly handling workloads without crashing.

- **High load**: this workload is designed to be above the system capacity for handling workloads by producing requests more frequently. The system should utilize adaptation logic to survive and provide its services to the users.

We measured the system response time and success rate for each stress test iteration. The application was first tested with no adapation to get the baseline numbers. Next the decision making model was configured to use an adaptation technique employing only mandatory goals specified as action policies processed by the ABLE policy engine. Subsequently the decision making model was configured to use an adaptation technique employing mandatory goals and negotiable goal clusters. Lastly, the decision making model was configured to use an adaptation technique based on mandatory goals and adaptive negotiable goal clusters. Two reinforcement learning algorithms namely Q-Learning and SARSA were used for the adaptive negotiable goal clusters. We used two different algorithms to examine the impact of using off-policy and on-policy algorithms respectively.

To minimize the experimental errors due to sporadic events, three replications were conducted for each stress test iteration. The specification of the machine used to conduct the experiments was: Windows Server 2003 Standard x64 Edition SP2, Intel Core 2 Quad CPU Q6700 @ 2.66GHz, 8GB of RAM.

5.2.5 Obtained Results

Figures 5.7, 5.8, 5.9, 5.10 and 5.11 display the obtained response times and success rates of the CC2 application when subjected to the two workloads. The obtained results clearly demonstrate the expected i.e. when the workload experienced by the CC2 application increases, the response time increases and the success rate decreases.

Moreover both the response time and success rate of the non-self-adaptive version of the CC2 application increase and decrease by a significant amount compared to its self-adaptive counterparts. Additionally, the results demonstrate that the overall response time of a decision-making model configured to use an adaptation technique based on mandatory goals and adaptive negotiable goals is better than or comparable to that of a decision making model configured to use an adaptation technique employing mandatory and negotiable

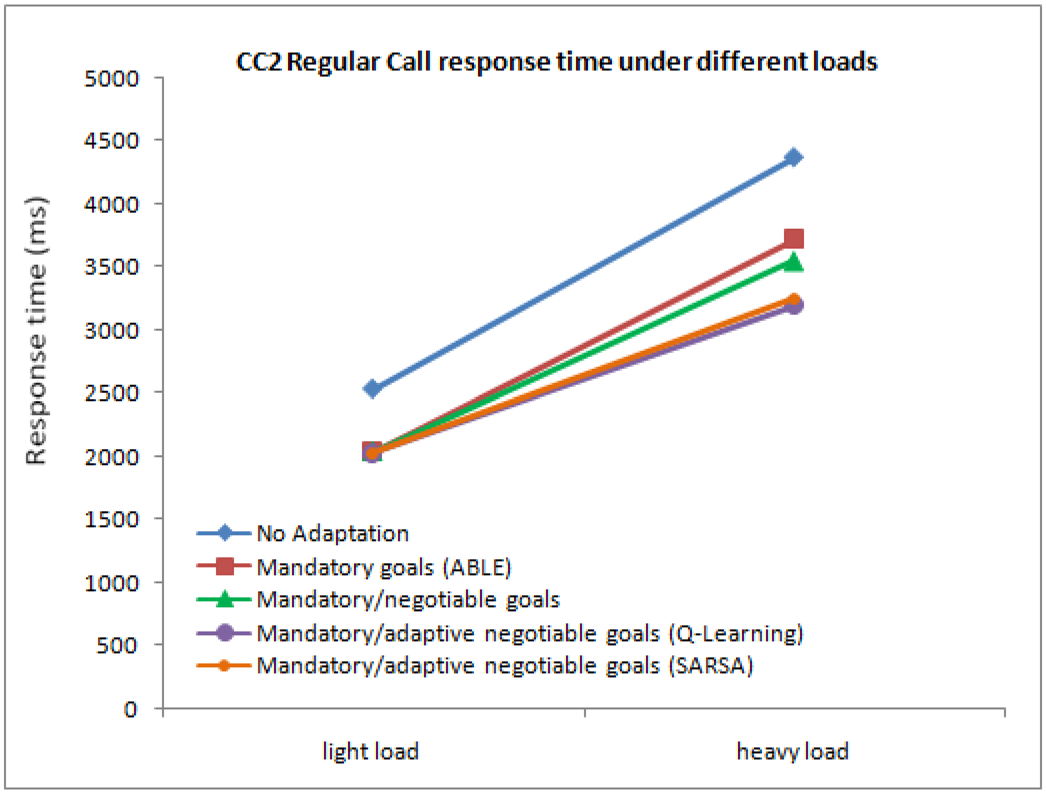


Figure 5.7: CC2: Regular Call Response Time

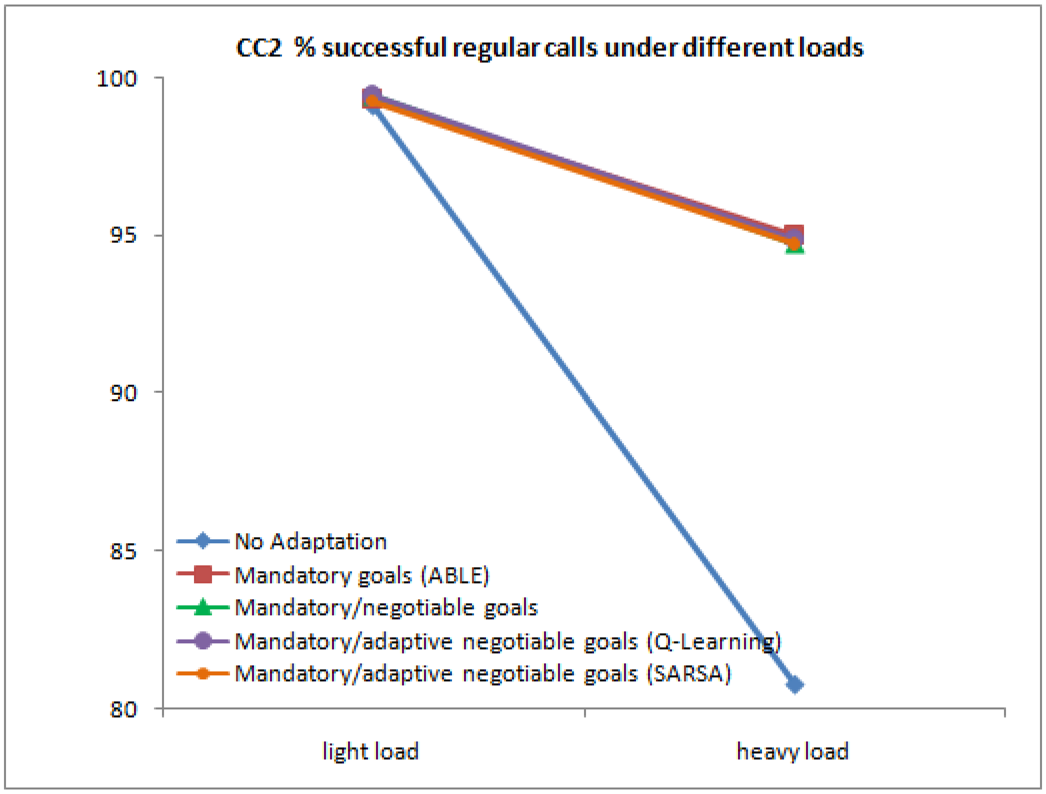


Figure 5.8: CC2: Percentage of Successful Regular Calls

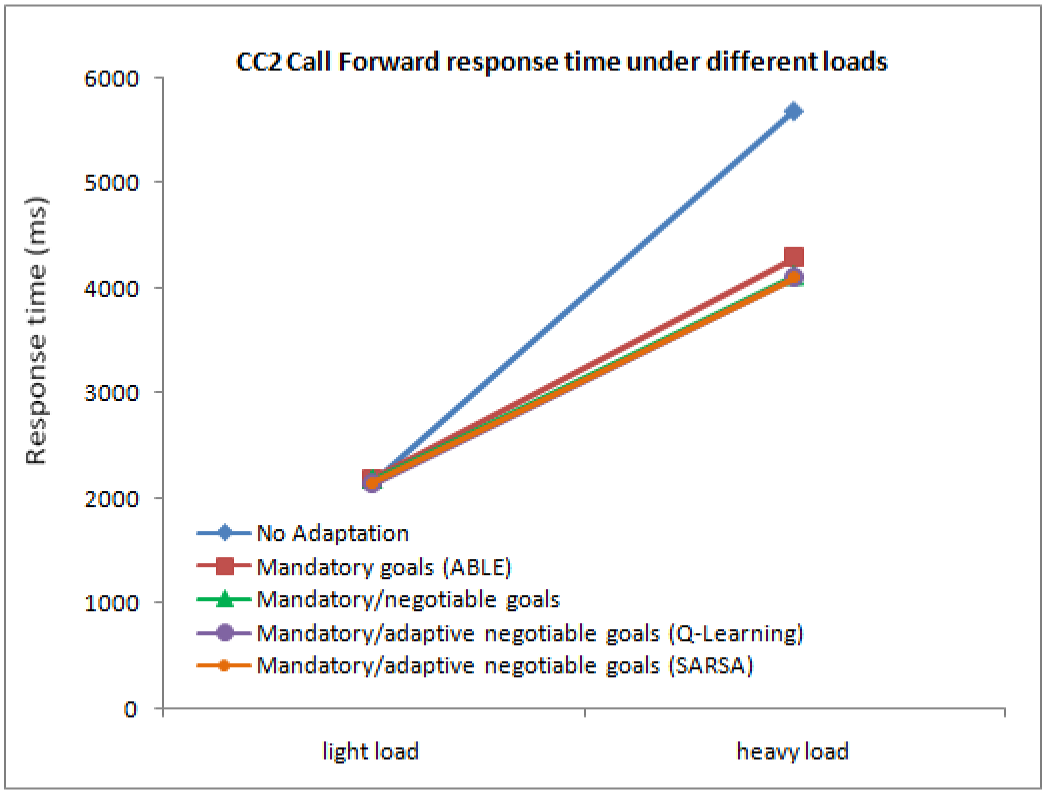


Figure 5.9: CC2: Call Forward Response Time

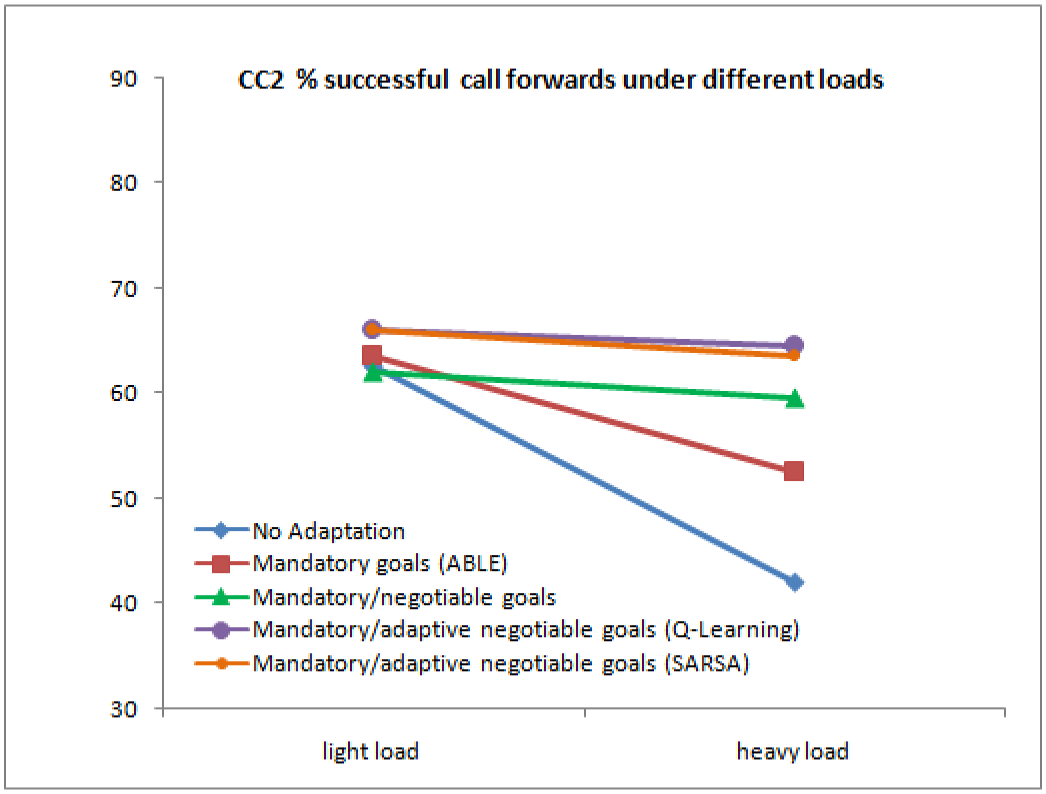


Figure 5.10: CC2: Percentage of Succesful Call Forwards

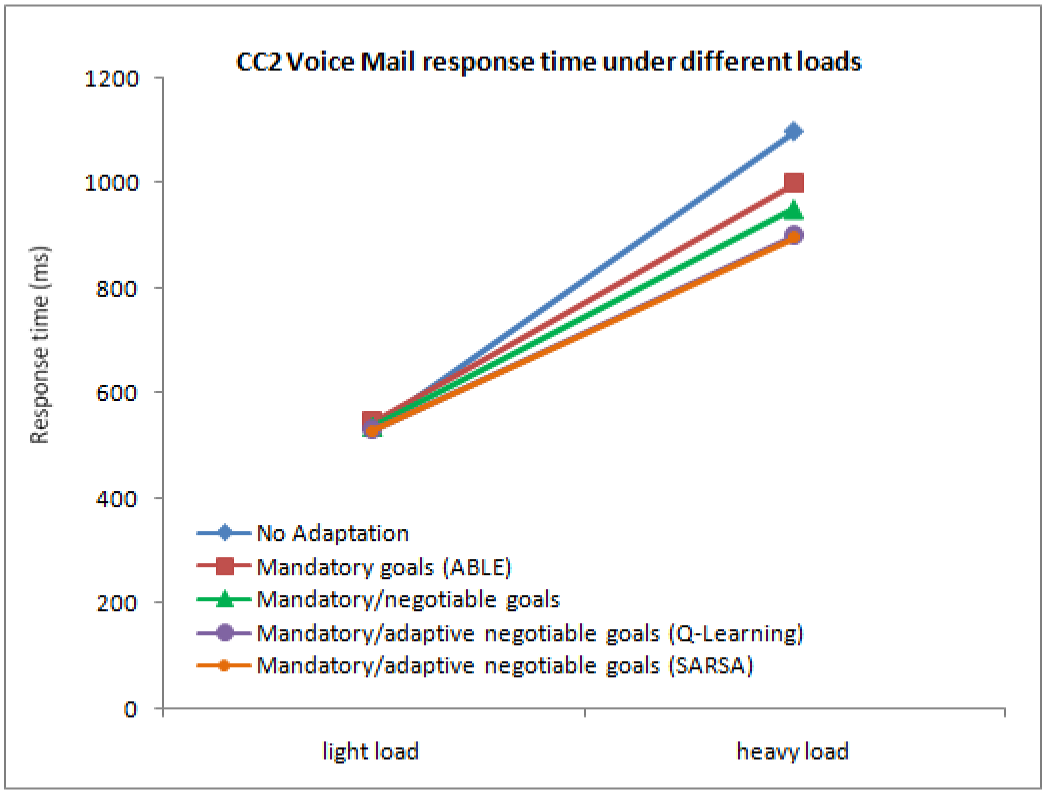


Figure 5.11: CC2: Voice Mail Response Time

goals or of a decision-making model configured to use an adaptation technique employing only mandatory goals specified as action policies processed by the ABLE policy engine.

Furthermore, the results obtained from the regular call subsystem demonstrate that the overall success rate of any self-adatpive version of the CC2 application is similar and is better than the overall success rate of the non-self adaptive version of the CC2 application.

Lastly, the results obtained from the call forwarding sub-system demonstrate that the overall success rate of a decision-making model configured to use an adaptation technique employing only mandatory goals specified as action policies processed by the ABLE policy engine is better than but similar to the overall success rate of the non-self-adaptive version of the CC2 application, where the success rate decreases by a sharp amount when the system is subjected to a higher workload. Based on the results we can also infer that the overall success rate of a decision making model configured to use an adaptation technique based on mandatory goals and adaptive negotiable goals is very similar to that of a decision making model configured to use an adaptation technique employing mandatory and negotiable goals.

The next section describes the overall comments on the results obtained from all the experiments conducted on both the case studies.

5.3 Lessons Learned

The objective of this chapter was to determine the impact of the adaptation techniques developed in this thesis on the system goals, and to determine the impact of making the control loop a first class entity in the decision making model as compared to other adaptation techniques. The adaptation requirements of the first case study i.e. the news web application were high service availability, maximum throughput and a reasonable response time. The adaptation requirements of the second case study i.e. the CC2 application were high service availability and a reasonable response time.

The results obtained from the experiments conducted on both case studies clearly demonstrated that the adapatation requirements were completely satisfied by all self-adaptive versions of the applications using the decision-making model developed in this thesis. Moreover these self-adaptive versions not only were in full compliance with these requirements, but also obtained significantly better results than their non-adaptive counterparts. On this bases, it cam be safely concluded that a self-adaptive system based on the decision-making model developed in this thesis improves the overall goal satisfaction level compared to the non-adaptive case.

Furthermore, the results obtained from the tests performed on both case studies show that a decision-making model based on mandatory goals and adaptive negotiable goals i.e.

a model where a control loop is treated as a first class entiity in a decision-making model can outperform a decision making model based solely on mandatory goals implemented as action policies or a model based on mandatory and negotiable goals. In cases where the model does not outperform the other models, it performs as well as the other models and thus does not lead to performance degradation.

Hence, we conclude that the decision-making model developed in this thesis shows promise and can certainly be used as a solution to overcome a large number of research gaps pertaining todecision-making models, that have been outlined in the previous chapters.

5.4 Summary

In this chapter we have presented an evaluation of the decision-making model developed in this thesis. The model has been evaluated using two case studies namely a news web application and an IP telephony system. Since both of the case studies were non-adaptive, we first converted the applications into self-adative applications using the guidelines mentioned in the previous chapters. We then evaluated the applications by subjecting them to a series of different workloads and observing the system attributes under those conditions. Both the non-adaptive and various self-adaptive versions of the applications were subjected to those tests. The self-adaptive versions used decision making models configured to use mandatory goals based on action policies executed using ABLE, mandatory and negotiable goal clusters, and mandatory and adaptive negotiable goal clusters. Based on the obtained results, it was concluded that the proposed decision-making model does improve the overall goal satisfaction level compared to the non-adaptive case. Moreover it was also concluded that a decision-making model based on mandatory and adaptive negotiable goal clusterss can outperform or perform as well as a decision-making model based exclusively on mandatory goals/action policies or a decision-making model based on mandatory and negotiable goal clusters.

Chapter 6

Conclusions and Future Direction

In this chapter, we summarize the findings of the thesis and outline future directions that can be pursued from this research. We begin this chapter by presenting the contributions of the research peformed during this thesis. Next, we outline some potential future work for extending thisresearch. In the last section we present some concluding remarks.

6.1 Contributions

The major contribution of the proposed research is that we have engineered a generic, configurable and enhanced goal-oriented decision-making model, that provides a comprehensive representation of all categories of adaptation requirements, enables representation of related flexible requirements as clusters, and also includes a mechanism for incorporating feedback control loops as first class entities in the decision making process. The principle contributions of this thesis were described in Chapter 1. We restate these with more information based on the remainder of the thesis:

- **Comprehensive Representation of Adaptation Requirements**: The decision-making model has been engineered such that it can be used to represent different categories of adaptation requirements ranging from mandatory to negotiable requirements.
- **Concurrent Satisfaction of Multiple Unrelated Adaptation Requirements**: The decision-making model has been designed to enable representation of related flexible adaptation requirements as clusters. It also provides support for the representing multiple clusters. Consequently the model provides ability to concurrently execute multiple corrective actions and thus satisfy mutiple unrelated adaptation requirements simultaneously.

- **Incorporation of Feedback Control Loops as First Class Entities**: The decision-making model enables the incorporation of feedback control loops as first class entities in the decision-making process of a self-adaptive system. This enables assessing the impact of a previously executed decision, so that better decisions can be made in the future, thus allowing the model to cope with run-time changes.

- **Conflict Detection and Resolution**: The decision-making model provides a mechanism to detect and resolve conflicts between dependent adaptation requirements.

- **Ultimate Flexibility in Specification of Voting Algorithm**: The decision-making model has been engineered to be extremely flexible. It provides the ability to specify any voting algorithm to choose a winner amongst competing flexible requirements. The realization of the model contains some built-in voting algorithms. However, a user is free to develop any voting algorithm that implements a specified interface.

- **Ultimate Flexibility in Specification of Reinforcement Learning Algorithm**: The decision-making model has been engineered to be extremely flexible. It provides the ability to specify any reinforcement learning algorithm to asses the impact of a previously executed decision, so that better decisions can be made in the future. The realization of the model contains some built-in reinforcement learning algorithms. However, a user is free to develop any reinforcement learning algorithm that implements a specified interface.

The next section describes the future research that can be conducted to enhance the proposed decision-making model.

6.2 Future Research Directions

As mentioned before, this work itself is an evolution of the GAAM model developed by Salehie et. al and there are still numerous ways to extend and improve this work. The following outline several potential directions in this area:

- **Automated Configuration File Generator**: Currently, the decision-making model's XML-based configuration file needs to be developed manually. This can be a tedious and error prone process if the system contains a large number of adaptation requirements. Hence, a future enhancement for this model would be the development of an accompanying graphical tool (e.g. an Eclipse-based plugin) that can be used to auto configure and generate an XML file based on some user input. This tool can also be leveraged to specify the StarMX configgruation information as well.

- **Graphical Monitoring Tool**: Another potential enhancement would be the development of a graphical tool that can be used to monitor the top level entities of the decision-making model and the underlying graph that is traversed by the decision-making process while determining the action to be executed to restore normalcy to the system.

- **Incorporation of Multi-Level Control Loops**: The decision-making model developed in this thesis does represent feedback loops as top level entities in the self-adaptative system by providing the ability to specify reinforcement learning algorithms to learn the preference values of the goal actions associated with an adaptive negotiable goal. However the priority of the goal itself is static for now. Providing the ability to learn the top level goal priorities leveraging machine learning techniques can be viewed as another potential research direction for this work.

- **Advanced Conflict Detection and Resolution**: At present the decision-making model uses statically assigned priorities to resolve conflicts between goal actions. Augmenting the model with the ability to include state of the art conflict detection and resolution algorithms can be viewed as another potential research direction. Moreover, augmenting the model with the ability to detect conflicts between goal definitions can also be viewed as a future research topic.

6.3 Conclusion

In this thesis we have engineered a generic, configurable and enhanced goal-oriented decision-making model, that provides a comprehensive representation of all categories of adaptation requirements, enables representation of related flexible requirements as clusters, and also includes a mechanism for incorporating feedback control loops as first class entities in the decision making process.

We began this thesis by presenting a literature review of research related to the different adaptation techniques employed by decision-making models in self-adaptive systems, discussed the merits and drawbacks of each appraoch and identified the research gaps in this area. We then presented an overview of the proposed decision-making model and its conceptual architecture. Then we described a procedure for building an adaptation manager consisting of the proposed decision-making model and StarMX which is a generic adaptation framework that can be used by Java-based self-adaptive systems. Next, we described a six-step process for developing a self-managing system using an adaptation manager composed of the proposed decision-making model and StarMX. Subsequently, we provided a description of the experimental studies conducted to evaluate the effectiveness of our model. We used two case studies namely a news web application and an IP telephony system and then presented an analysis of the obtained results.

Based on the obtained results, it was concluded that the proposed decision-making model does improve the overall goal satisfaction level compared to the non-adaptive case. Moreover it was also concluded that a decision-making model based on mandatory goals and adaptive negotiable goal clusters can outperform or perform as well as a decision-making model based exclusively on mandatory goals/action policies or a decision-making model based on mandatory and negotiable goal clusters.

APPENDICES

Appendix A

Sample Configuration File for the News Web Application

The following shows a sample configuration file for the news web application:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE gaam PUBLIC "-//STAR Lab//GAAM Configuration DTD//EN" "gaam.dtd">
<gaam>
  <!-- attributes -->
  <attribute
      attribute-id="att_main_servlet_url"
      mbean-id="MainServletMBean"
      mbean-attribute-name="ServletURL"
      mbean-attribute-type="String" />
  <attribute
      attribute-id="att_main_servlet_hostname_port"
      mbean-id="MainServletMBean"
      mbean-attribute-name="HostNameAndPort"
      mbean-attribute-type="String" />
  <attribute
      attribute-id="att_main_current_load"
      mbean-id="MainServletMBean"
      mbean-attribute-name="CurrentLoad"
      mbean-attribute-type="Integer" />
  <attribute
      attribute-id="att_main_resp_time"
      mbean-id="MainServletMBean"
```

```
    mbean-attribute-name="AverageResponseTime"
    mbean-attribute-type="Integer" />
<attribute
    attribute-id="att_main_throughput"
    mbean-id="MainServletMBean"
    mbean-attribute-name="AverageThroughput"
    mbean-attribute-type="Double" />

<attribute
    attribute-id="att_news_display"
    mbean-id="NewsMBean"
    mbean-attribute-name="MediaTypeAttributeValue"
    mbean-attribute-type="String" />

<attribute
    attribute-id="att_stock_display"
    mbean-id="StocksMBean"
    mbean-attribute-name="DisplayStocks"
    mbean-attribute-type="String"/>

<attribute
    attribute-id="att_weather_display"
    mbean-id="WeatherMBean"
    mbean-attribute-name="DisplayWeather"
    mbean-attribute-type="String"/>

<!-- actions -->
<action
    action-id="act_news_display"
    action-type="atomic"
    mbean-id="NewsMBean"
    mbean-attribute-name="MediaTypeAttributeValue"
    mbean-attribute-type="String"/>

<action
    action-id="act_main_servlet_url"
    action-type="atomic"
    mbean-id="MainServletMBean"
    mbean-attribute-name="ServletURL"
    mbean-attribute-type="String"/>
```

```
<action
    action-id="act_main_servlet_hostname_port"
    action-type="atomic"
    mbean-id="MainServletMBean"
    mbean-attribute-name="HostNameAndPort"
    mbean-attribute-type="String"/>
<action
    action-id="act_main_servlet_init"
    action-type="composite">
      <sub-action action-id="act_main_servlet_url" />
      <sub-action action-id="act_main_servlet_hostname_port" />
</action>
<action
    action-id="act_main_servlet_stop_service"
    action-type="atomic"
    mbean-id="MainServletMBean"
    mbean-attribute-name="StopService"
    mbean-attribute-type="Boolean"/>

<action
    action-id="act_stock_display"
    action-type="atomic"
    mbean-id="StocksMBean"
    mbean-attribute-name="DisplayStocks"
    mbean-attribute-type="String"/>

<action
    action-id="act_weather_display"
    action-type="atomic"
    mbean-id="WeatherMBean"
    mbean-attribute-name="DisplayWeather"
    mbean-attribute-type="String"/>

<!-- goals -->

<!--
Initially set the servlet URL and servlet host name and port
to the appropriate value to start the internal monitoring
process within the servlet.
-->
```

```
<mandatory-goal
    goal-id="set_main_servlet_url"
    priority="10"
    activation-function-operation-type="and" >
    <goal-attribute
        attribute-id="att_main_servlet_url"
        threshold-val-isnull="true"
    />
    <goal-attribute
        attribute-id="att_main_servlet_hostname_port"
        threshold-val-isnull="true"
    />
    <goal-action
        action-id="act_main_servlet_init">
        <goal-action-props mbean-attribute-value=
               "http://127.0.0.1:8080/servlet-war/servlet/main" />
        <goal-action-props mbean-attribute-value=
               "127.0.0.1:8080" />
    </goal-action>
</mandatory-goal>

<!--
Goal - The system should always be up and running.
Translation - Control the maximum number of requests
received by the system such that if the requests exceed
a threshold return a 404 page instead of returning a
servlet exception.
-->
<mandatory-goal
    goal-id="system_up_and_running"
    priority="9">
    <goal-attribute
        attribute-id="att_main_current_load"
        threshold-max="100" />
    <goal-action
        action-id="act_main_servlet_stop_service">
        <goal-action-props mbean-attribute-value="true" />
    </goal-action>
</mandatory-goal>
```

```
<!--
Goal1 - The system should have a minimum response time.
Translation -  Control the image quality and display
options to guarantee the designated minimum response
time.
Goal2 - The system should provide maximum throughput.
Translation - Control the image quality and display
options to guarantee the designated maximum throughput.
Goal3 - The sytem should provide the best UI experience
i.e. display the highest quality information.
Transaction - If highest quality information is not being
displayed then try and do so.
-->
<adaptive-negotiable-goal-cluster
  cluster-id="c1"
  priority="8"
  voter-type="borda-count"
  rl-agent-type="q-learning" >
  <rl-agent-properties>
      <rl-agent-property
                  property-name="Q_LEARNING_ALPHA"
                  property-value="0.3" />
      <rl-agent-property
                 property-name="Q_LEARNING_GAMMA"
                 property-value="0.01" />
  </rl-agent-properties>
  <adaptive-negotiable-goal
    goal-id="goal1_1"
    priority="40"
    activation-function-operation-type="and" >
    <goal-attribute
        attribute-id="att_main_current_load"
        threshold-max="30"
        weight="0.5" />
    <goal-attribute
        attribute-id="att_main_resp_time"
        threshold-max="2000"
        weight="0.5" />
    <goal-action
        action-id="act_news_display"
        preference="3">
        <goal-action-props mbean-attribute-value="BETTER" />
    </goal-action>
    <goal-action
        action-id="act_news_display"
        preference="2">
        <goal-action-props mbean-attribute-value="GOOD" />
    </goal-action>
  </adaptive-negotiable-goal>
  <adaptive-negotiable-goal
    goal-id="goal1_2"
    priority="50"
    activation-function-operation-type="and" >
    <goal-attribute
        attribute-id="att_main_current_load"
        threshold-max="45"
        weight="0.01" />
    <goal-attribute
        attribute-id="att_main_resp_time"
        threshold-max="3000"
        weight="0.99" />
    <goal-action
        action-id="act_news_display"
        preference="4">
        <goal-action-props mbean-attribute-value="GOOD" />
    </goal-action>
    <goal-action
        action-id="act_news_display"
        preference="3">
        <goal-action-props mbean-attribute-value="NONE" />
    </goal-action>
    </goal-action>
  </adaptive-negotiable-goal>
  <adaptive-negotiable-goal
    goal-id="goal1_3"
    priority="60"
    activation-function-operation-type="and" >
    <goal-attribute
        attribute-id="att_main_current_load"
        threshold-max="55"
```

```
        weight="0.05" />
    <goal-attribute
        attribute-id="att_main_resp_time"
        threshold-max="4000"
        weight="0.95" />
    <goal-action
        action-id="act_news_display"
        preference="3">
        <goal-action-props mbean-attribute-value="NONE" />
    </goal-action>
    <goal-action
        action-id="act_stock_display"
        preference="2">
        <goal-action-props mbean-attribute-value="false" />
    </goal-action>
    <goal-action
        action-id="act_weather_display"
        preference="1">
        <goal-action-props mbean-attribute-value="false" />
    </goal-action>
  </adaptive-negotiable-goal>
 </adaptive-negotiable-goal-cluster>
</gaam>
```

References

[1] S. Liaskos A. Lapouchnian, Y. Yu and J. Mylopoulos. Requirements-driven design of autonomic application software. In *Proc. International Conference on Computer Science and Software Engineering*, 2006. 15

[2] R. A. Van Lamsweerde, R. Darimont and P. Massonet. Goal-directed elaboration of requirements for a meeting scheduler: problems and lessons learnt. In *Proc. of the Second IEEE International Symposium on Requirements Engineering*, 1995. 15

[3] R. Asadollahi. Starmx: A framework for developing self-managing software systems. Masters Thesis, Univeristy of Waterloo, 2009. 55, 65

[4] J. Lobo D. Agrawal, C. Seraphin K. Lee and D. Verma. Policy technologies for self-managing systems. IBM Press, 2008. 13

[5] B. H. Cheng D. M. Berry and J. Zhang. The four levels of requirements engineering for and in dynamic adaptive systems. In *Proc. of 11th International Workshop on Requirements Engineering: Foundation for Software Quality*, 2005. 15

[6] A Elkhodary and S. Malek. Fusion: A framework for engineering self-tuning self-adaptive software systems. In *International Symposium on the Foundations of Software Engineering (FSE)*, 2010. 10, 17, 18, 25, 26

[7] Wikipedia: The Free Encyclopedia. Q-learning. Wikimedia Foundation Inc, http://www.wikipedia.org, 2011. 22, 23, 24

[8] Wikipedia: The Free Encyclopedia. Reinforcement learning. Wikimedia Foundation Inc, http://www.wikipedia.org, 2011. 22

[9] A. Mukhija et al. Runtime adaptation of applications through dynamic recomposition of components. In *Proc. of Int. Conf. on Architecture of Computing Systems*, pages 124–138, 2005. 10, 11, 17, 18

[10] B. H. Cheng et al. Software engineering for self-adaptive systems. In *Lecture Notes in Computer Science*, 2009. 1, 4, 5

[11] D. Garlan et al. Rainbow: Architecture-based selfadaptation with reusable infrastructure. In *IEEE Computer*. 10, 11, 12, 17, 18

[12] G. Kaiser et al. An approach to autonomizing legacy systems. In *Workshop on Self-Healing, Adaptive and Self-MANaged Systems*, 2002. 3, 10

[13] G. Tesauro et. al. A hybrid reinforcement learning approach to autonomic resource allocation. In *Proc. of the 2006 IEEE International Conference on Autonomic Computing*, 2006. 24, 26

[14] H. Liu et al. A component-based programming model for autonomic applications. In *Proc. of International Conference on Autonomic Computing*, pages 10–17, 2004. 10, 11, 12, 18

[15] J. White et al. Simplifying autonomic enterprise java bean applications via model-driven development: A case study. In *Proc. of Int. Conf. on Model Driven Eng. Languages and Systems*, pages 601–615, 2005. 10, 11, 18

[16] L Lymberopoulos et. al. An adaptive policy based framework for network services management. In *Journal of Network and Systems Management*, 2003. 25, 26

[17] N. Dulay et al. A policy deployment model for the ponder language. In *Proc. of IEEE/IFIP International Symposium on Integrated Network Management*, 2001. 25, 26

[18] P. Vienne et. al. A middleware for autonomic qos management based on learning. In *Proc. of the 5th international workshop on Software engineering and middleware*, 2005. 25, 26

[19] R. M. Bahati et al. Modelling reinforcement learning in policy-driven autonomic management. In *IEEE/IARIA International Journal On Advances in Intelligent Systems*, pages 54–79, 2008. 10, 13, 14, 18, 25, 26

[20] S. M. Sadjadi et al. Trap/j:transparent generation of adaptable java programs. In *Lecture Notes in Computer Science*, pages 1243–1261, 2004. 10, 18

[21] H. Goldsby and B.H Cheng. Goal-oriented modeling of requirements engineering for dynamically adaptive system. In *Proc. of the 14th IEEE International Requirements Engineering Conference*, 2006. 15

[22] B. Henderson-Sellers and P. Giorgini. Agent-oriented methodologies. Idea Group Inc, 2005. 15

[23] M. G. Hinchey and R. Sterritt. Self-managing software. In *IEEE Computer*, pages 107–109, 2006. 1

[24] IBM. An artificial intelligence perspective on autonomic computing policies. In *Proc. of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, 2004. 12

[25] IBM. An architectural blueprint for autonomic computing. White-paper, http://www-01.ibm.com/software/tivoli/autonomic/pdfs/AC Blueprint White Paper 4th.pdf, 2006. 1

[26] J. R. Pilgrim W. N. Mills J. P. Bigus, D. A. Schlosnagle and Y. Diao. Able: A toolkit for building multiagent autonomic systems. pages 350–371. IBM Systems Journal, 2002.

[27] J. O. Kephart and D. M. Chess. The vision of autonomic computing. In *IEEE Computer*, pages 41–50, 2003. 2

[28] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In *Future of Software Engineering*, 2007. 16

[29] E. Kant L. Brownston, R. Farell and N. Martin. An introduction to rule-based programming. Addison-Wesley, 1985.

[30] A. Van Lamsweerde. Goal-oriented requirements engineering: a guided tour. In *Proc. of Fifth IEEE International Symposium on Requirements Engineering.* 9, 15

[31] E. Letier and A. Van Lamsweerde. Reasoning about partial goal satisfaction for requirements and design engineering. In *Proc. of ACM SIGSOFT Int. symposium on Foundations of software eng*, pages 53–62, 2004. 15

[32] E. C. Lupu and M. Sloman. Conflict analysis for management policies. In *Proc. of IFIP/IEEE International Symposium on Integrated Network Management*, pages 430–443, 1997.

[33] G. Pavlou A. K. Bandara E. C. Lupu A. Russo N. Dulay M. Sloman M. Charalambides, P. Flegkas and J. Rubio-Loyola. Policy conflict analysis for quality of service management. In *Proc. of IEEE International Workshop on Policies and Distributed Systems and Networks*, pages 99–108, 2005.

[34] L. Penserini M. Morandini and A. Perini. Towards goal-oriented development of self-adaptive systems. In *Proc. of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, 2008. 15

[35] E. Martins. jain slee example call-controller-2. http://code.google.com/p/mobicents, 2008.

[36] Oracle. Sun java management extensions. http://jcp.org/en/jsr/detail?id=3.

[37] Oracle. Sun jvm tool interface. http://java.sun.com/j2se/1.5.0/docs/guide/jvmti.

[38] R. N. Taylor P. Oreizy, N. Medvidovic. Architecture-based runtime software evolution. In *Proc. of the 20th international conference on Software engineering*, 1998. 17

[39] E.P. Kasten P.K. Mc Kinley, M. Sadjadi and B.H.C. Cheng. Composing adaptive software. In *IEEE Computer*, pages 56–64, 2004. 3, 8

[40] Robert R. Darimont and A. Van Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. In *SIGSOFT Software Engineering Notes*, 1996. 9, 15, 16

[41] S. Russell and P. Norvig. Artificial intelligence: A modern approach. Prentice Hall, 2003.

[42] M. Salehie. A quality-driven approach to enable decision-making in self-adaptive software. PhD Thesis, Univeristy of Waterloo, 2010. 29, 30

[43] M. Salehie and L. Tahvildari. A quality-driven approach to enable decision-making in self-adaptive software. In *Companion to the proceedings of the 29th International Conference on Software Engineering*, 2007. 16

[44] M. Salehie and L. Tahvildari. A weighted voting mechanism for actionselection problem in self-adaptive software. In *International IEEE Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 328–331, 2007. 3, 8, 10, 11, 16, 18, 31

[45] R. S. Sutton and A. G. Barto. Introduction to reinforcement learning. MIT Press, 1998. 21, 22

[46] G. Tesauro. Online resource allocation using decompositional reinforcement learning. In *Proc. of the 20th national conference on Artificial intelligence*, 2005. 25

[47] D. Vengerov and N. Iakovlev. A reinforcement learning framework for dynamic resource allocation: First results. In *Proc. of the Second International Conference on Automatic Computing*, 2005. 25, 26

[48] C. Gacek H. Giese H. Kienle M. Litoiu H. Muller M. Pezzè Y. Brun, S. Marzo and M. Shaw. Engineering self-adaptive systems through feedback loops. In *Lecture Notes In Computer Science*, 2009. 19