SELF-MANAGEMENT MIDDLEWARE SERVICES FOR AUTONOMIC GRID COMPUTING

Wail M. Omar

A thesis submitted in partial fulfilment of the requirements of Liverpool John Moores University for the degree of Doctor of Philosophy

February 06

Abstract

Over the coming years, many are anticipating global (grid) computing infrastructures, utilities and services to play a major role in expediting the realisation of the long overdue agile enterprise business model. For which global computing environments will be required to exhibit self-managing capabilities to reducing cost, complexity and improve dependability. The realisation of such a vision necessitates the addressing of a host of socio-technical concerns related to software provisioning, assurance and auditing including; cost of access and ownership, reliability, dependability, interoperability, ubiquity, security and complexity.

Many commentators contend that the IBM proposed autonomic computing vision will offer a crucial paradigm shift to delegating vital functions of self-management systems including: configuration, healing, tuning and protecting to the software itself, along with curbing the ever increasing complexity. A prevailing design model of autonomic computing systems is one of a model-based architecture governed by policies via rule-based meta-systems [1], which provide autonomic capabilities such as; self-configuration, self-optimisation, self-tuning, self-protective, self-organising, self-governance and/or self-healing. Whilst, such a rule-based approach has been successfully applied to support self-managing systems with inherently stable operating rules (and/or policies), however, more remains to be done to develop a fundamental understanding of standards, reference models and generic support for self-managing decentralised systems including an understanding of associated models for rules (policies) evolution and management.

Concerned with the autonomic software systems engineering, this work focuses on the development of a generic reference model and generative approaches for open standard self-managing global computing system. Such proposed models are underpinned by two existing models, namely; distributed object programming (middleware) and Viable System Model (VSM). In addition, the proposed model for the global computing considered the environment as a collection of clouds (zones), which provides a middleware partitioning abstraction and structuring of application services for improved control and management of client applications through customised cloud area policies.

Furthermore, this research presents the generic requirements for tools, services and frameworks to facilitate the design and development of self-management systems. For instance, the Assembly Services and Infrastructures Framework (ASIF) with its allied description language are developed and implemented to offer a fabric for utilising different types of resources in open standard format. Monitoring model has been designed in this research to provide the situated autonomic computing with feedback and context information, from their environment, including instrumentation and sensor data. In support of the monitoring and awareness services, a Sensor and Actuator Framework (SAF), together with its associated description languages, have been developed and evaluated both quantitatively and qualitatively using a range of scenarios applications including; On-Demand Services, E-health Monitoring System and Monitoring PlanetLab Environment.

Acknowledgements

At the outset, I would like to express my appreciation to Professor A. Taleb-Bendiab from the School of Computing and Mathematical Sciences, Liverpool John Moores University for his advice during my doctoral research endeavour for the past three years. As my supervisor, he has constantly forced me to remain focused on achieving my goal. His observations and comments helped me to establish the overall direction of the research and to move forward with investigation in depth. I thank him for providing me with the opportunity to work with a talented team of researchers. I also would like to show gratitude to the director of the School Professor Madjid Merabti for his support during my stay at Liverpool John Moores University.

I would like to express my sincere thanks and gratitude to my father and mother for their continued support, and my sincere appreciation to my sister Huda and my brother Basil.

I greatly appreciate Dr. Naeem and Mrs. Widad for their help and assists in writing this thesis.

Last, but not least, I would like to dedicate this thesis to my family, my wife Asmaa, my daughter Aya and my son Faisal, for their love, patience, and understanding—they allowed me to spend most of the time on this thesis.

Wail M. Omar

Table of Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1.Motivations

Driven by modern needs for open standard, dependable and interoperable computing infrastructure, the e-business community has been instrumental in the convergence of a number of technologies namely; web services [2], service-oriented architecture [3], semantic web [4], grid computing [3] and autonomic computing [5]. In particular, web services standards and models are continuously extended through the necessity to underpin fast evolving modern business models characterised by global scale virtual organisations – Agile Enterprises, for which the next generation Internet is anticipated to provide overlay mechanisms upon existing global computing infrastructures providing end-users programming, interaction and control models to develop, deploy and manage their required applications in a timely and seamless way. In other word, the next generation of Internet will take further steps beyond the information sharing service that is available now reaching to resources sharing.

However, such a seductive vision comes at a heavy price bringing with it many technical challenges (Sec. 1.2). Many of which are already under investigation including: new theories, computational paradigms, languages and implementation techniques for the design, development, deployment and management of future global computing environments. Specific to this work, the autonomic computing community is exploring and developing models to support lifetime management of distributed systems' by delegating many of the systems management, maintenance tasks to the software itself including [6]; resource management, services failure prediction, load-balancing, QoS, services reservations, and resources deployment and discovery.

A prevailing design model of autonomic computing systems is one of a model-based architecture governed by policies via rule-based meta-systems [1], which provide autonomic capabilities such as; self-configuration, self-optimisation, self-tuning, self-protective, self-organising, self-governance and/or self-healing. Whilst, such a rule-based approach has been successfully applied to support self-managing systems with inherently stable operating rules (and/or policies). However, more remains to be done to develop a fundamental understanding of open standards, reference models and generic supports for self-managing decentralised systems including an understanding of associated models for rules (policies) evolution and management. In addition, self-management middleware service for autonomic computing capabilities is addressed as a vital for producing survival distributed enterprise applications.

1.2.Challenges

Self-management software services are imperative to manage the planetary-scale system in order to offer high QoS, reliability, resources availability and security. To achieve these goals, many theoretical and practical challenges need to be addressed in order to achieve ultimately a comprehensive model of systems' self-management. These challenges can be addresses as follows:

- **Reference models:** including software design patterns, baseline architecture and/or middleware for developers to design, develop, deploy, manage and monitor self-management system. In doing so, there is a need to take into account the inherent uncertainty, complexity and scalability issues related to the planetary-scale systems.

- **Mechanisms:** including utilities for structuring and configuring distributed components in order to form a trusted environment that can provide guaranteed services to the consumers. To this end, a variety of services, tools and a framework need to be designed, developed and implemented providing a set of utilities including:

- Normative description of self-management policies and strategies for sensing, deliberation and actuations.

- Shared interoperation model and protocols for the interchange of systems' metamodel between different actors.
- Self-management intelligence and anticipatory behaviour to support predictable system self-adaptation ensuring stability and survivability.
- Refining self-management rules via machine learning.
- Testbeds to demonstrate the autonomic computing capabilities, services, tools and framework which are corporate together to form an intelligent unit that can be embedded into any system in order to carry out self-management services.
- **Interoperation standards:** including markup languages to support the design of open standard self-managing systems.

1.3.Research Hypothesis

To insure the dependability requirements of service-oriented networked applications, while coping with their design heterogeneity, intrinsic complexity and responsive adaptation to unpredictable network conditions, a novel set of autonomic middleware services and self-management reference architectural model are required.

Thus, this work's research hypothesis is based on the assumption that well-established system theory, software design patterns and semantic middleware services, which can provide a foundation for the design, development and management of autonomic global computing system.

1.4.Approach

The work described in this thesis aims to investigate the generic requirements for self-management system for planetary-scale environment development. Moreover, it attempts to describe services, tools, and frameworks required to run autonomic computing capabilities in order to achieve the vision of the self-management system. For theoretical support this work draws on a number of research results emerging from related fields including:

- **Advanced software engineering:** using middleware services to bridge the gap between the network layer and the application layer. The services offered by the middleware cover discover, deploy and invoke services.
- **Self-management systems:** using proposed models, requirements and theories to enable software to use real-time monitoring, diagnosis, repair and control [7] systems to sort out an automated mechanism for managing the environment according to the nature and boundaries of applications.

In addition, this work in nature follows an experimental research approach, which starts by a proposal of a new architectural model of self-managing systems, which is then rigorously designed, implemented and evaluated using a number of case examples. Throughout the process the proposed model is iteratively refined leading to a generalise set of requirements, pattern language, framework and middleware services.

1.5.Contributions

This work makes a number of novel contributions towards a better understanding of planetary-scale self-management system requirements and architectural models, the detail of which can be found in Chapter 10, and have been (or are being) peer reviewed for publications [8-22]. These contributions can be summarised as follows;

- **Cloud-Based Model for Planetary-Scale System:** which provides an organisation model for widely distributed systems into hierarchical structures (clustering or zoning) to partition and localise systems control and management behaviour along self-similarity principles [23, 24]. Such model assists in containment and overcoming the inherent systems complexity due to heterogeneity, scalability, the chaotic systems and instability behaviour which is often triggered by self-management processes.
- **Self-Management Architecture and Support:** This provides autonomic systems reference model including middleware supports. For instance, they are employed to carry out the automated and intelligent behaviour required to perform the self-management activities and improve the performance and

ability of the system in the sense of response to different changes in environment behaviours. This contribution also covers the description of the interference between varieties of autonomic computing capabilities. In addition, this work describes the design, development and implementation of autonomic computing capabilities based on using intelligent services.

- **Interoperability Support:**
  - *Services and infrastructures metamodel:* to standardised and systematize the deployed information that is employed to describe the usability of resources. The developed metamodel is vital to discover a common language for interoperable information between the resources providers and consumers. Moreover, this metamodel offers the required information for managing the resources container in order to provide better QoS, fidelity and availability.
  - *Distributed monitoring framework:* to assist the planetary-scale middleware as well as its applications to audit, manage and control the resources in order to track any malfunction in the functions of the system. The monitoring system depends on planetary-scale overlay in order to provide the required resources for gathering information and perform actions in the monitored targets.
  - *Sensors and actuators metamodel:* to structure a semantic way for deploying, discovering and invoking monitoring resources. Such metamodel provides the monitoring system with the demanded information for performing the required tasks of selecting and using monitoring resources.
- **Self-Managing Software Design Pattern:** To achieve the above goals, this research is built on middleware core services including: deployment, discovery and invocation together with more advanced services, including:
  - *The assembly service:* containing the service description, configuration, parameters and execution functionalities.

- *The monitoring system:* containing the deploying, discovering and invoking of variety of sensors and actuators, which provide monitoring and actuation capabilities.
- *Resources container:* to include all types of resources from services and infrastructures that deployed with the environment.

1.6.Scope

This research has proposed a new self-management reference model to specify and design autonomic distributed application. This model is build on Internetworking five layers model (application, transport, network, data-link and physical layers) with an adding new layer (middleware layer) to support the deploy, discover, invoke and manage the planetary scale resources. The middleware layer is proposed to support the cloud/zone abstraction in order to partition control and management of virtualised software applications services.

To achieve this model, this work focuses on:

- The development and modifying of the Internet model in order to suitable for the planetary-scale environment and large-scale enterprise applications.
- The development of a generic model for self-management service in order to specify the life-cycle of requesting self-management capabilities by the environment as well as its applications.
- The development of a services and infrastructures framework that offers a number of middleware activities which are required to serve the ultimate goal of self-management system. These activities are deploy, discovery, invocation monitoring, recovery, management, etc.
- The development of a service description model that assists the client to obtain significant information from the resources container in order to reduce the response time by increasing the fidelity in selecting the required resources.
- The development of a monitoring framework that provides a fabric for deploying, discovering, invoking and managing monitoring resources from

sensors, actuators and analysers. Such developed framework depends on planetary-scale overlay for providing monitoring resources.

- The development of monitoring description model that helps the consumers (users/ applications and providers) to deploy and discover variety types of sensors and actuators depending on standard parameters.
- The design and development of description model for requesting monitor session taking in consideration variety of information regarding the duration of collecting readings, targets, application and requested resources. This description model assists the consumers in finding standard method for requesting monitor resources.
- The design, development and implementation of intelligent services as web service in order to be acceptable in the planetary-scale system. The research focuses on adopting more than one technique like machine learning and mathematic algorithm to carry out the intelligent stuff.
- The merger of more than one autonomic computing capability to offer new structure of the autonomic capabilities that offers comprehensive functions and features.

The approach for developing the generic model described in this thesis has been tested in three different enterprise applications, namely: on-demand services, E-Health Monitoring System, and Monitoring PlanetLab Environment. The developed models, algorithms, services, tools, and frameworks are tested through these examples (Sec. 9.4).

1.7.Thesis Structure

This thesis consists of ten chapters and is organised as follows:

- Chapter 1, introduces the main motivations of the work, challenges, contributions and thesis outline.
- Chapter 2, introduces the relevant background theories, principles and technologies relevant to this work. This covers the basic concepts and

principles of planetary-scale system through taking grid computing in details as an example of such environment. Grid computing components, capabilities, architectural models, topologies and Open Grid Service Architecture have been presented in this chapter. Moreover, PlanetLab environment is outlined here, which will be used in Chapter 9 as an additional evaluation case of planetary-scale system.

- Chapter 3, provides the relevant background for the self-management system and autonomic computing capabilities including relevant standards and reference models for autonomic computing.

- Chapter 4, reviews the state-of-the-art and related work covering a range of fields namely: self-management distributed systems, reference models, autonomic computing applications and capabilities, self-management middleware services and existing distributed monitoring system. Finally, the chapter ends with an outline of the main requirements for the proposed middleware services and associated frameworks.

- Chapter 5, describes the overall requirements for designing self-management model as well as autonomic computing models for planetary-scale systems. Autonomic computing capabilities for planetary-scale system are also presented in this chapter.

- Chapter 6, starts by a proposed extension of the OSI network model to support the new requirements of the global computing systems. This will be followed by a detailed description of a self-managing software pattern design which is based on combination of the managerial design pattern of the Viable System Model and the Gang of Four software design patter.

- Chapter 7, demonstrates the required support utilities for self-management middleware service depending on using autonomic computing. Intelligent web service design, services and infrastructures framework, and sensor and actuator framework are described in this chapter.

- Chapter 8, documents the required description languages to support self-management system. Assembly Services and Infrastructures Description Languages, Sensor and Actuator Description Language and Monitoring Session Description Language are outlined in this chapter.
- Chapter 9, describes the evaluation of this work; gives the assumptions of the previous chapters and uses example applications such as on-demand service, E-health monitoring system and monitoring PlanetLab environment.
- Chapter 10, presents a summary, achievement and contribution, concluding remarks and proposed future work.

CHAPTER 2

PLANETARY-SCALE SYSTEM

2.1.Introduction

A Planetary-scale (or global computing) system integrates networking, communication, computation and information to provide a virtual platform for widely decentralised enterprise applications [6, 25-28]. This chapter outlines the basic concepts of planetary-scale system through a detailed description of grid computing including; the basic components of grid computing, and architectural grid models. Moreover, grid topologies are classified in this chapter into intraGrid, extraGrid and interGrid. Furthermore, Open Grid service Architecture (OGSA), which is the convergence of grid technology with web service technology, has been introduced in this chapter. At the end, Planetary-scale overlay is also presented.

2.2.Grid Computing

Over the coming years, many are anticipating that grid computing infrastructures, utilities and services to become an integral part of future socio-economical fabric. The realisation of such a vision will be very much affected by many factors including; cost of access, reliability, dependability and security of grid computing services. Hoschek [29] defined grid computing as;

> *"... collaborative distributed Internet systems characterized by large scale, heterogeneity, lack of central control, multiple autonomous administrative domains, unreliable components and frequent dynamic change ... ".*

Whereas, Berman *et al.* [3] defined grid computing as;

"...The Grid is the computing and data management infrastructure that will provide the electronic underpinning for a global society in business, government, research, science and entertainment..."

From the above definitions, the benefits of grid computing to support enterprise business application are accrued through collaborative distributed resources and information sharing including; software, hardware and associated content, to build one large system serving all subsystems and consumers. On the other hand, such computing models engenders many research problems and concerns that need to be addressed including; heterogeneous and decentralized system, dynamic infrastructure, sharing of resources, security issues, management issue, using different network/connection protocols, and absence of a common data representation.

The following sections describe grid computing components, capabilities, architectures and topologies.

2.2.1. Grid Computing Components

There are numbers of common components that are proposed be available in every type of grid computing models and architectures [30]. These components are:

- **Middleware:** to enable the interoperation of heterogeneous distributed systems running on different platform and operating system. In addition, middleware is in charge of interoperable information between different actors of the environment. Moreover, middleware offers a fabric for monitoring, controlling and managing the usability of resources within environment, taken in consideration the level of security of the system. There are different types of middleware as shown in Appendix A. In this research, .Net is adopted as a middleware for deploying our services, tools, frameworks and infrastructures. The reason for selecting .Net is its ability in dealing with web services in the sense of deploying, discovering and invoking.
- **Network Communication and Protocols:** to facilitate the interaction and liaising of information from one node to another node. This includes cables (guided and unguided media), routers, bridges and other network devices. In

addition, this covers the required protocols from Network Model [31] that insures the transfer of messages between the actors of the system.

- **Services:** to offer a layer of services that can be utilised by the consumers in order to carry out their activities. Services cover all type of business, commercial, government, health, sciences and other types of services. Moreover, services provide the required dependences for running the enterprise applications.

- **Infrastructure:** to offer a fabric of resources for running distributed enterprise application. Such resources cover supercomputer, clusters, software, storage systems, and others.

- **Consumer:** this is expressed as a user, application and/or auditing system. The consumer is expected to get remote services from the grid computing environment in order to execute their demands.

2.2.2. Grid Computing Capabilities

The following describe the important capabilities of grid computing that assist in clarifying the expected usability of such technology.

- **Exploiting Resources:** As Ferreira *et al.* [32] pointed out that the ideal use of grid computing is to run distributed application on different machines. The machine on which the application is normally run might be busy due to the peaks in activity; therefore, it could be run on an idle machine elsewhere on the grid.

- **Resources Allocation:** availability, reliability, interpretability and Service Level of Agreement (SLA) still need a lot of investigations to demonstrate the grid computing competence in performing the process of exploring resources. Resources allocation competence assists the enterprise application in increasing its fidelity in discovering the required resources.

- **Parallel Processing:** The potential for massive parallel processing is one of the most attractive features of a grid computing [33]. Such computing power

is driving a new evolution in industries like financial modelling, motion picture animation, and many others that can be partitioned into independently running parts to reduce the time of processing by splitting the application among many CPUs.

- **Running Large-Scale Enterprise Applications:** There are many factors to be considered in grid-enabling an application. One must understand that not all applications can be transformed to run in parallel on a grid and achieve scalability according to the fact that not all the programs can be partitioned [32]. Therefore, the programming direction of using Web services in grid computing is increasing rapidly [34] because web services are considered as classes that can be distributed over the grid environment to get more processing power [35]. The deploying and using of large-scale enterprise application through grid environment require more investigations to address the needs for achieving QoS, reliability, security and intelligence (automated) of such applications.

- **Virtual Resources Collaboration:** Another important capability of the grid computing contribution is the facilitating and simplifying the collaboration among a wider audience to give better services. In the past, distributed computing promised this collaboration and achieved it to some extent [36]. However, grid computing is often presented as the next step towards resources virtualisation and sharing for the wider community. This assumption characterised large virtual computing systems offering a variety of virtual resources.

- **Resource Balancing:** Grid computing consists of a huge numbers of resources from services, infrastructures and networking collaborate together in order to offer reliable services to the consumers with high performance [32]. Thus, resources load balancing is required to enhance the utilisation of grid computing in terms of resources availability, reliability and QoS [36].

- **Reliability:** Ferreira *et al.* [32] described the reliability of the grid computing resources as *"...High-end conventional computing systems use expensive*

hardware to increase reliability...". The reliability of the resources is discussed from two perspectives; a hardware and networks failure and software services failure viewpoints. The next gain in building reliable systems is now focused on software and software services reliability and resilience. Therefore grid computing emerged to address [33] the development of low cost high-performance, high-reliable and high-availability computing.

- **Management:** The goal of the resources availability on the grid is to disperse information technology's infrastructure and handle heterogeneous systems [32]. In such heterogeneous, decentralised and distributed information, administrators (consumers and/or control system) provide the system with policies, rules and strategies that handle and manage how the different organisations might share or compete for the resources after getting real-time information from the web. In addition, grid computing environment is expected to manage the resources in a way that improve the critical parameters such as; reliably, fidelity, QoS and others.

- **Open Standards:** The idea is to convince the community of software engineers currently developing the grid, including those from major IT companies, to set common standards for the grid up-front. The open standards community assists the applications to communicate with infrastructures, services and tools in formalization and semantic ways in addition to offer a way for describing the use of resources in standard format.

2.2.3. Grid Architecture Models

It should be mentioned that grid technologies are still in the development and implementation period but in general, there are different types of grid architectures has been proposed to meet the needs of variety of business applications [32]. The following sections describe the common models of the grid computing.

2.2.3.1. Computational Grid

As listed in the grid capabilities above, computational grid aggregates the processing power from a distributed collection of systems (Fig. 2.1). Ferreira *et al.* [32]

categorized the computational grids by a set of primary characteristics, which can be summarised as:

- Made up of "clusters of clusters", or in another word clouds or zones.
- Enables CPU scavenging in order to provide better utilised resources.
- Provides the computational power to process large scale jobs.
- Satisfies the business requirement for instant access to resources on-demand.
- Offers a fabric of resources to provide reliable and high performance system.

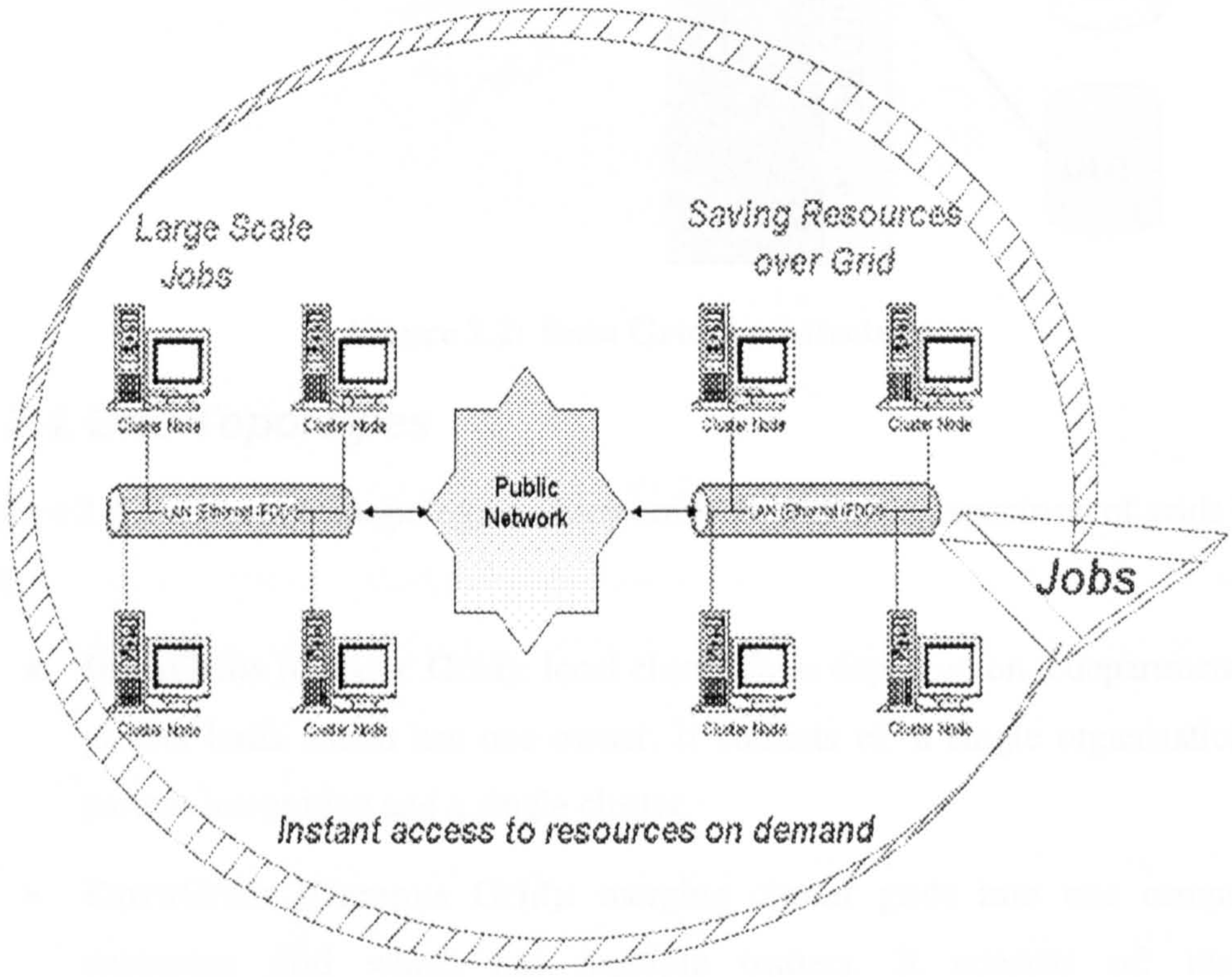


Figure 2.1: Structure of Computational Grid [32]

2.2.3.2. Data Grid

Ferreira *et al.* [32] defined the Data grids as the way of providing secure access to distributed and heterogeneous databases. Grid computing offers a fabric for deploying and accessing different types of database in semantic format based on utilising XML. The big database engines like Oracle, SQL Server, My SQL and others have started

implementing database grid. Figure 2.2 demonstrates accessing grid database by the consumer depending on well known standards like SOAP, OLEDB, ODBC and others.

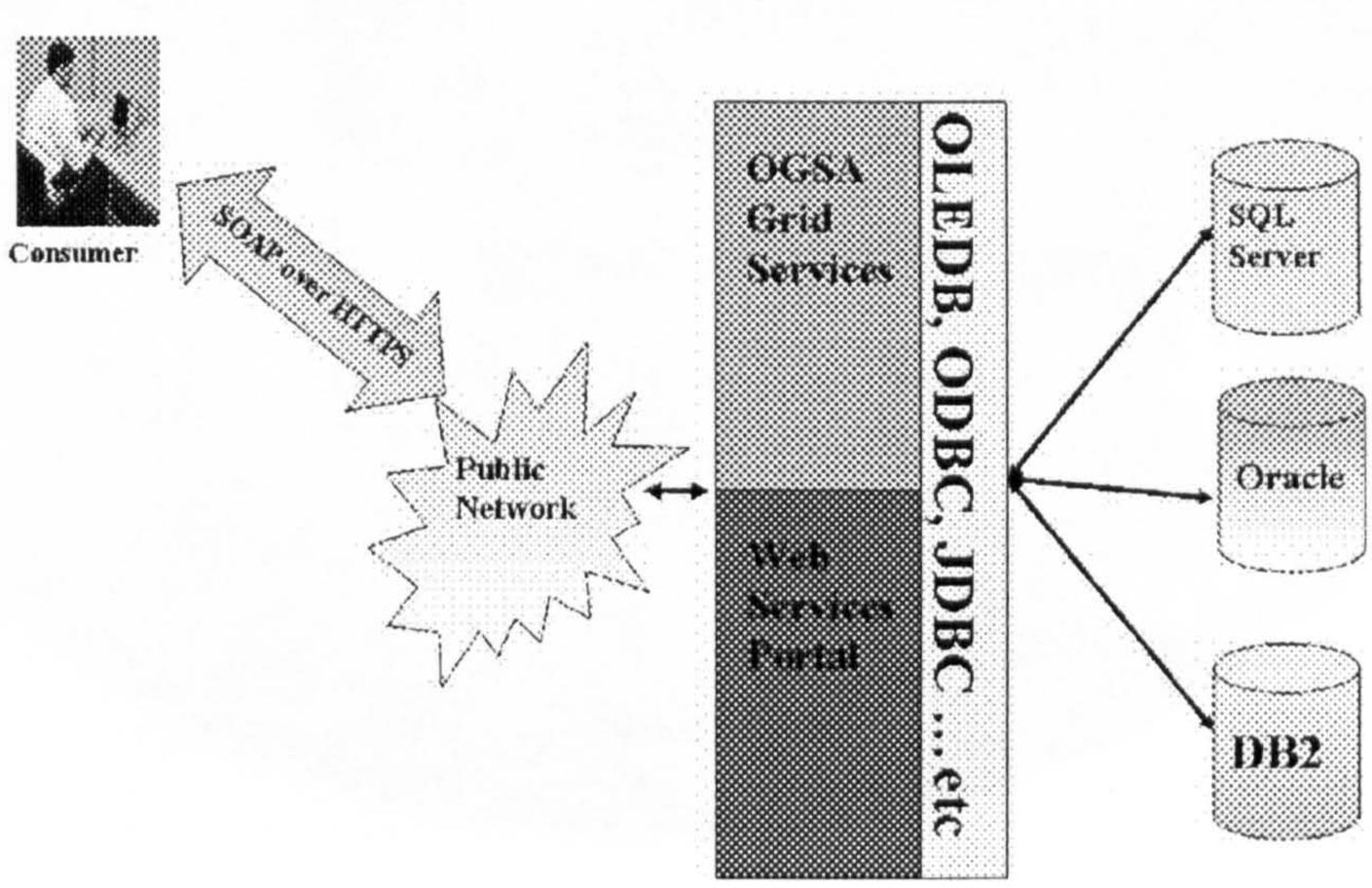


Figure 2.2: Data Grid Architecture

2.2.4. Grid Topologies

Figure 2.3 illustrates the topologies which cover the following spectrum of grids [3, 4, 32]:

- **IntraGrids (Cluster Grid):** local cluster/farm deployed on a departmental or project basis which has one owner. It consists of: a single organisation, no partner integration and a single cluster.
- **ExtraGrids (Campus Grid):** merging cluster grids into one campus or enterprise grid which has multiple owners. It consists of: multiple organizations, partner integration and multiple clusters.
- **InterGrids (Global Grid):** merging campus grids into multiple projects and global grid across organization which has multiple sites. It consists of: many organizations, multiple partners and many multiple clusters

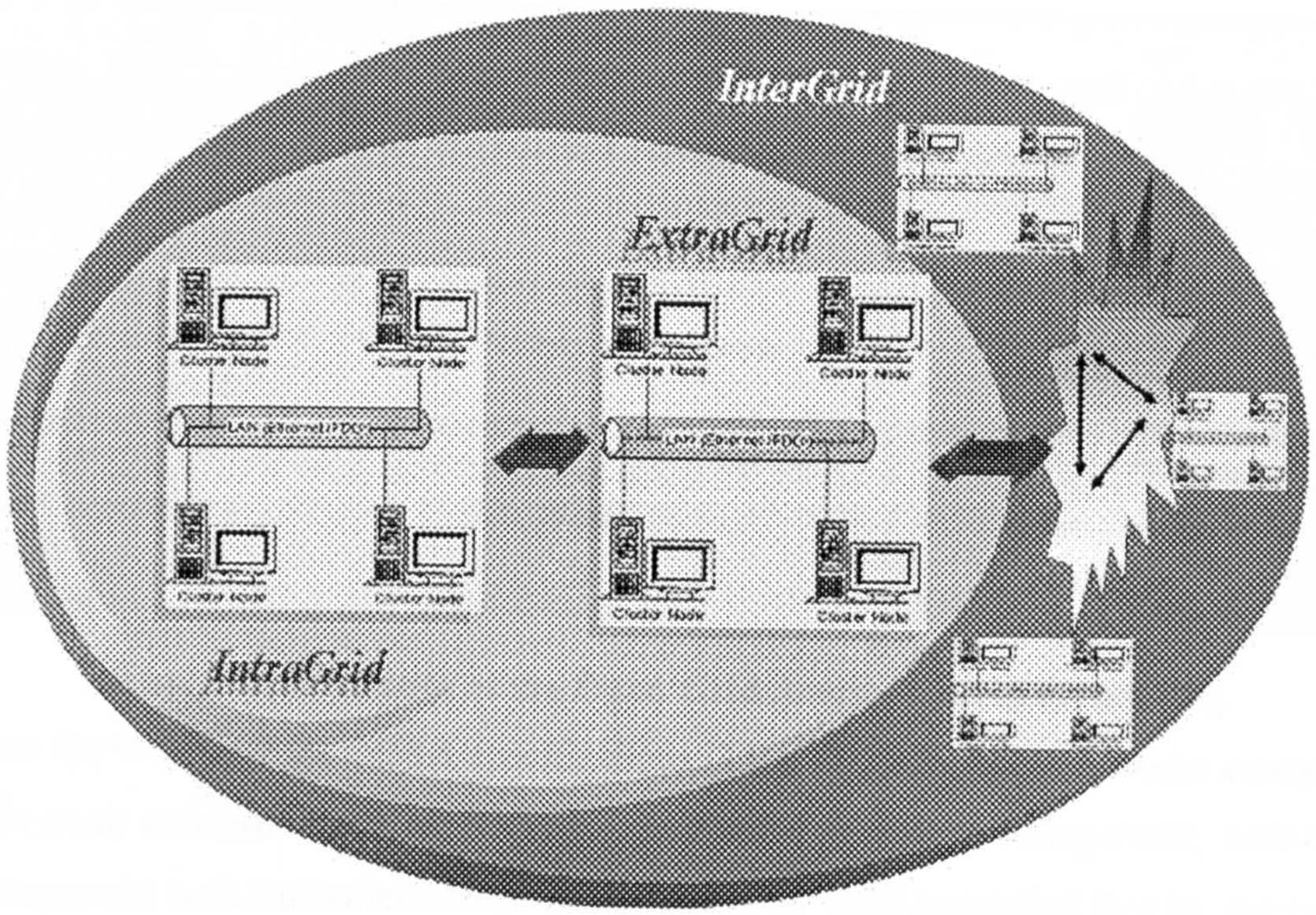


Figure 2.3: Grid Topologies

2.3.Open Grid Service Architecture

IBM [35] defines the Open Grid Services Architecture (OGSA) as:

> *"...a distributed interaction and computing architecture that is based around the Grid service, assuring interoperability on heterogeneous systems so that different types of systems can communicate and share information...".*

OGSA is a convergence of grid computing technology with Web Services Technology and standards [3, 4]. In that, grid computing provides the specification for distributed computation using shared managed resources. While, the web services specification provides an open standard distributed object and/or service-oriented programming model. Hence, OGSA specification bridges the gap between grid service definition (interface) and service hosting discovery and activation.

Figure 2.4, illustrates the OGSA framework structured into a five layered stack namely: applications, OGSA, web services, auditing and services [35].

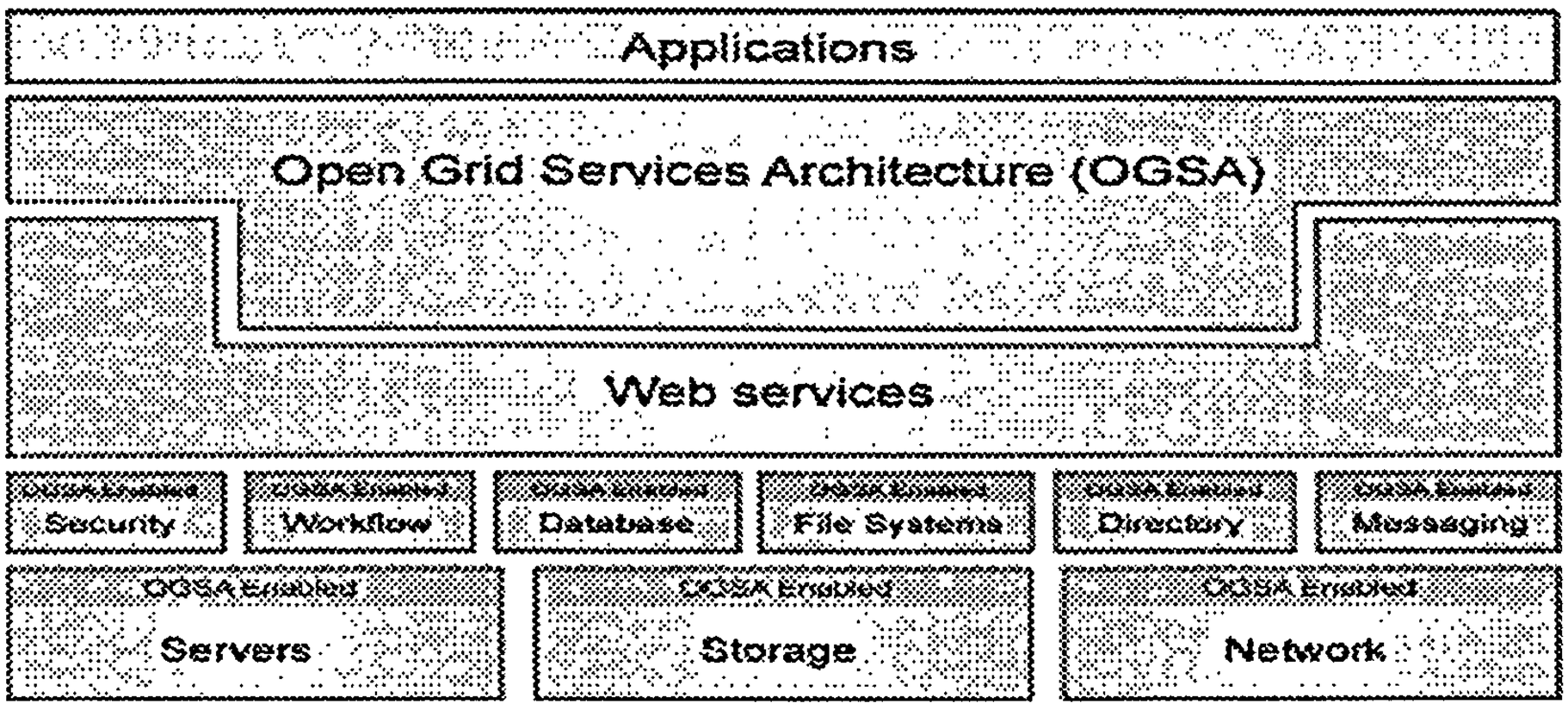


Figure 2.4: Structure of the Open Grid Services Architecture [35]

Since the release of the OGSA specifications [35] a range of research works emerged addressing concerns such as; security, distributed resource management, network management and fault-tolerance [36]. Ferreira *et al.* [32] suggested that by merging the grid with the web services technologies, the OGSA increases the data grid capability in several ways, which are:

- Files or databases can seamlessly cover many systems and thus have larger capacities than for any stand alone systems. Such range or spanning can improve data transfer rates via the use of striping techniques.
- Data can be mirrored or replicated throughout the grid to other host in order to support the backup algorithms and techniques.
- Sharing is not restricted to files/directories, but also might contain many other resources, such as devices, software, services, licenses, and others resources. These resources are used to give a more uniform interoperability among heterogeneous grid consumers. The consumers of the grid can be part of several actual and virtual organizations. The grid can assist in applying security rules among the organizations and implement policies, which can assign priorities for both resources and consumers.

2.4.Planetary-Scale Overlay

PlanetLab is an open platform testbed originally set up by Intel etc. to provide a research testbed for developing Internet, WAN network and global computing research [37-39]. PlanetLab model depends on location partitioning or zoning where each region has a central unit known as a Principal Investigator (PI). The latter is responsible for all the management, control and monitoring processes of the different components in the considered region. Each of which can aggregate a number of computer hosts (nodes), slices, slivers, Virtual Servers (VServer), infrastructures, communications, applications and users [38, 39].

Hence, as detailed in Section 9.4.3, the PlanetLab testbed was used to develop sensor and actuator overlay for autonomic global computing systems. The PlanetLab currently distributed over 629 nodes published on 298 sites [38]. The PlanetLab components are [38, 40]:

- **Site:** is a physical location where PlanetLab nodes are running.
- **Node:** is a dedicated server that runs components of PlanetLab services.
- **Slice**: is a set of allocated resources distributed across the PlanetLab overlay. To most users, a slice means a UNIX shell access to a number of PlanetLab nodes. After being assigned a slice, a user may then assign nodes to it, and then virtual servers for that slice are created on each of the assigned nodes. Slices have a finite lifetime and must be periodically renewed to remain valid.
- **Sliver:** is a set of allocated resources on a single PlanetLab node.
- **Virtual Server (VServer):** is a Linux server that separates the user-space environment into distinct units (sometimes called Virtual Private Servers) in such a way that each VPS looks and feels like a real server to the processes contained within. Moreover, VServer supports resource limited with Class-based Kernel Resource Management (CKRM) [41] , and virtually networked with VNET.

- **Principal Investigator (PI):** is an actor that is responsible for managing slices and users at each site. PIs are legally responsible for the behaviour of the slices that they create. Most sites have only one PI.
- **Technical Contact (Tech Contact):** is an actor responsible for installation, maintenance, and monitoring of the site's nodes.
- **User:** is anyone who develops and deploys applications on PlanetLab. PIs may also be users.

2.5.Summary

As a summary of what has been discussed in this chapter, planetary-scale systems are expected to support large-scale enterprise applications. Grid computing and OGSA are employed in order to improve the high-reliability, QoS, and availability along with reduction the cost of ownership. To achieve these goals, many services and infrastructure should be used with this framework. Such resources should be managed well depending on automated way to reduce the interaction of consumers in order to improve the feasibility of the use of distributed applications. Next chapter descries the autonomic computing technology which should be adopted and merged with planetary-scale environment in order to offer self-management services. The autonomic computing services should be designed and developed as one of the middleware functions in order to inherit the security and control polices from the middleware. Moreover, autonomic computing middleware services should depend on global computing overlay to perform the expected self-management services.

CHAPTER 3

SELF-MANAGEMENT SYSTEM

3.1. Introduction

Engendered by modern global economy, agility and dependability imperatives for decentralised enterprise applications require new methods to support their development and management. This amongst many other drivers has been a motivating factor to explore biologically-inspired computational models to deal with complexity, non-determinism, heterogeneity and uncertainty inherent in the design and lifetime management of large-scale distributed software.

In particular, autonomic computing research is exploring and developing models to hide the complexity and reduce administration costs of software systems by delegating many of the systems' management and maintenance tasks to the software itself including; resource management, job scheduling, services failure prediction, load-balancing, QoS, resources allocation, services reservations and discovery. A prevailing design model of autonomic computing systems is one of the models and goal-based approach, where the rules are developed through a given domain analysis, data mining and/or heuristics based rules.

This chapter introduces autonomic computing model including: standards, architectures and key capabilities such: self-healing, self-tuning, self-configuration, self-optimising, self-protective and self-organising. All of these capabilities depend on elaborate mechanisms to imbue a given system's with accurate introspection (awareness) of the self and its environments, and a perceptual interpretation and identification of the need to react in a controllable manner to ensure safety and predictability of systems' behaviour and impact on its environment.

3.2.Autonomic Computing

Over the past two decades, whilst research and development yielded much progress towards continuous improvements of ICT's cost, reliability, security, adaptation to change [5, 42, 43]. However, a recent nature-inspired paradigm shift is now driving research towards imbuing systems with capabilities to self-manage [44].

Self-management depends on a number of factors that specify the policies, strategies and rules to achieve the required tasks. For example, self-management utilities can organise a system execution to provide an optimum fault recovery plan, or execution scheduling for improved QoS. The following sections provide definitions and descriptions of the core self-management capabilities.

3.3. Definitions

Ganek and Corbi [45] defined the autonomic computing as

> *"...Autonomic computing represents a collection and integration of technologies that enable the creation of an information technology computing infrastructure for the next era of computing—e-business on demand..."*

Murch [46] defined the successful autonomic systems as:

> *"... The systems that have the ability to manage themselves and dynamically adapt to change in accordance with business policies and objectives, enabling computers to identify and correct problems often before they are noticed by IT personnel... "*

Tosi [47] defined the autonomic computing system as:

> *"... The systems that able to limit hands-on intervention in case of exceptional situations (such as unexpected workload variation, or failure in the system, or introduction of a new component, or new interactions among components, or external attacks), and the system must be able to modify own behaviour to adapt to changes that may compromise functional or non-functional correctness... "*

3.3.1. Autonomic Computing Characteristics

As indicated in [48, 49] the five defining characteristics of an autonomic system can be summarised as follows:

- **Introspection and self-awareness:** where autonomic system requires reflective and grounding ability to be aware of self and non self, and distinguish between normal and abnormal behaviour including its structural, behavioural models and identity.
- **Situatedness:** where autonomic system acts in accordance and react to its environment.
- **Intentionality:** where autonomic system deliberates and acts in a goal-driven model to for instance to reconfigure to adapt to any unexpected malfunction.
- **Immunity:** An autonomic computing system must be an expert in protection itself from unexpected attacks. Moreover, it must detect, identify and protect itself against various types of insecure behaviour in order to superintend the system to safe situation.
- **Optimisation:** An autonomic computing system anticipates the optimised resources needed for keeping its complexity hidden from the user in that implementation. [50, 51].

3.3.2. Autonomic Computing Capabilities

Many published works have identified and in many cases studies a range of autonomic computing capabilities [46, 47, 51-53], which can be described as follows:

- **Self-Healing capability:** which is defined by Tosi [47] as "*...a system's capability to examine, find, diagnose and react to system malfunctions...*". Hence, as proposed by Badr *et al.* [54, 55] to ensure predictability and governance of self-healing systems, their reaction should be controlled by policy (or norm) specification [45]. In addition, self-healing process consists of a number of phases starting by ***monitoring*** a given systems behaviour, which is compared to stored normal system behaviour model [47]. If a

deviation or anomaly (such as failure) is observed. This will be followed by a ***diagnosis*** of the symptoms and ***planning*** the course of action (repair strategies). Then ***enacting*** and validating the repair plan otherwise the self-healing process starts again.

- **Self-Protective capability:** which is a system's mechanisms to detect, identify, anticipate and/or react to protect itself from internal [31] or external attacks [45] or mal-function. Chess *et al.* [50] defined the goal of self-protective capability as: *"... The ability of a system to react consistently and correctly to situations ranging from benign but unusual events to outright attacks... "* [50, 56]. Typically, a self-protective capability is responsible for managing the authentication, authorization and accounting in accordance with a given Service Level of Agreement (SLA).

- **Self-Configuration capability:** Which facilitates autonomous dynamic change of a system's structural model for instance to add, remove or replace a set of components in response to a self-healing process [51, 56]. As indicated in [54, 55, 57] both triggering and affecting the self-configuration should be governed (regulated) by a policies [47]. For example, in case of overloaded system, self-configuration capability is in charge of specifying the policies of embedding new system in order to perform load balancing technique.

- **Self-Optimising capability:** Which facilitates a system to monitor and autonomously optimise its operation such as resources utilisation, performance, or other [45, 46]. Self-optimising capability depends on the use of a variety system's metrics and optimisation policies. For example, if the reliability of the system is the key parameter, then self-optimising service organises the system to ensure the availability of the service [46]. Thus, self-optimising should be integrated with other capabilities in order to specify the objectives of the running environment.

3.3.3. Autonomic Computing Standards

The development of autonomic computing standards is acknowledged as one of the most important enabler to leveraging the development, deployment and lifetime management of autonomic services in the heterogeneous planetary-scale system. Such a standard should facilitate the integration and interoperation of cost autonomic components. The United States government's National Institute of Standards and Technology (NIST) [58] notes:

> *"... Standards are essential elements of information technology-hardware, software, and networks. Standard interfaces, for example, permit disparate devices and applications to communicate and work together. Standards also underpin computer security and information privacy, and they are critical to realizing many widespread benefits that advance in electronic and mobile commerce are anticipated to deliver..."*

Representative standards used by an autonomic manager are described as four functions that share knowledge system [42, 51, 59-61], which are:

- **The monitor function:** to provide the mechanisms for collecting, aggregating, filtering and reporting details collected from a managed resource.
- **The analyse function**: to provide the mechanisms that correlate and model complex situations.
- **The plan function**: to provide the mechanisms that construct the actions needed to achieve goals and objectives. The planning mechanism uses policy information to guide its work.
- **The execute function**: to provide the mechanisms that control the execution of a plan with considerations for dynamic updates.

As shown in Figure 3.1, these four functions work together to provide the control loop functionality [42, 51, 59-61]. In addition, the four functions communicate and collaborate with one another and exchange appropriate knowledge and data. Sensors are employed in this model to collect information from the system or environment.

Effectors are used to carry out an action that is demanded by the autonomic computing services.

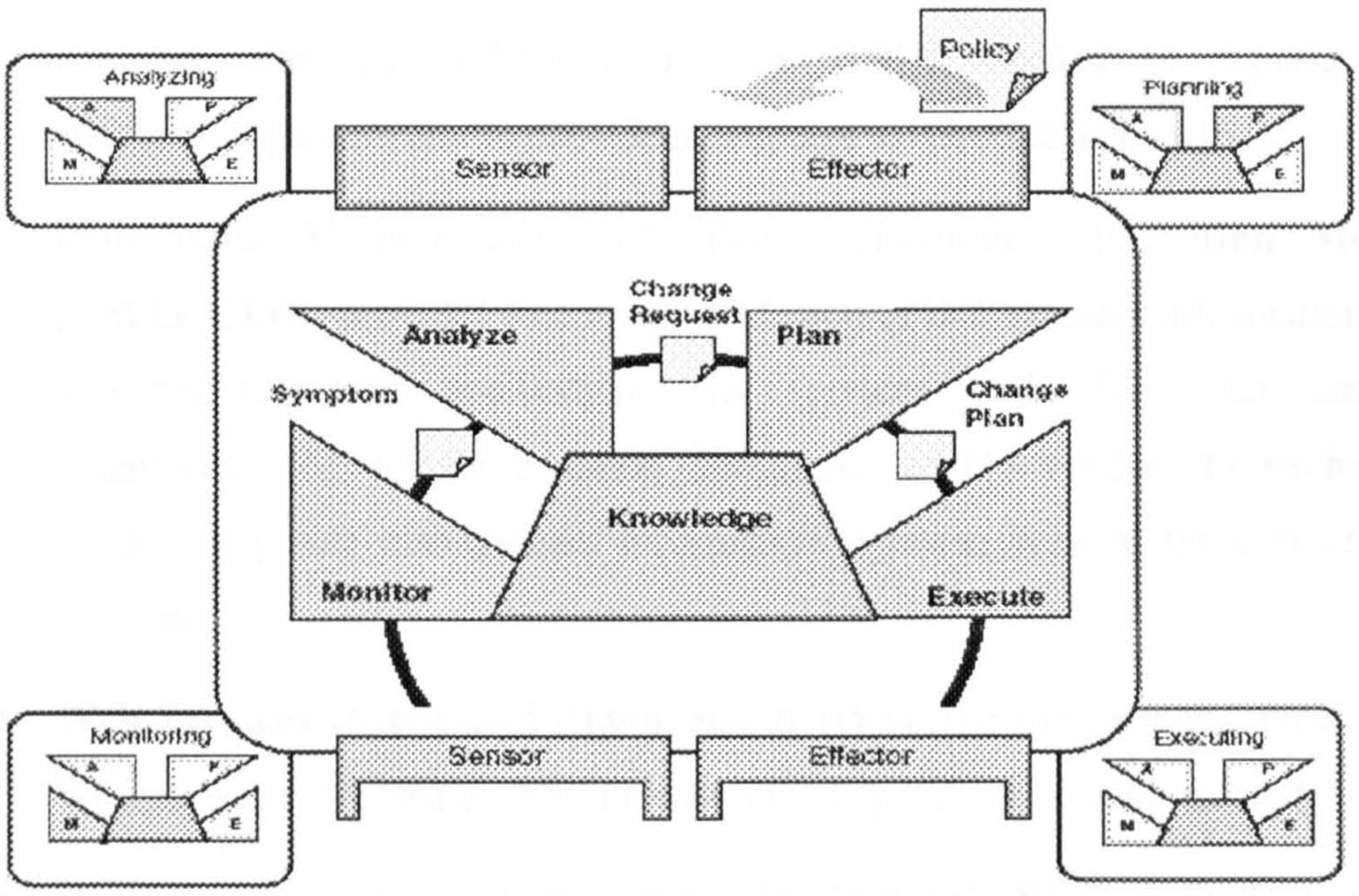


Figure 3.1: Functional Details of the Autonomic Manager [46]

3.3.4. Autonomic Computing Interoperability Standards

As shown in Figure 3.2, Miller [42] outlined a set of interoperability standards required for information exchange, thus open-standard design and management of autonomic software systems. The standards are outlined below:

- **Web Service Agreement (WS-Agreement):** defines a taxonomy and terminology including concepts, overall agreement structure with types of agreement terms, agreement template with creation constraints and protocols for creation, and negotiation and renegotiation of agreements [62] needed to request monitoring resources from sensors and effectors.
- **Web Service Policy (WS-Policy):** provides a flexible and extensible grammar for expressing the capabilities, requirements, and general characteristics of entities in an XML Web services-based system [63]. WS-

Policy defines a framework and a model for the expression of these properties as policies.

- **Web Service Notification (WS-Notification):** provides a terminology, concepts, message exchanges and the WSDL needed to express the notification pattern, and to provide a language to describe Topics [64].

- **Distributed Management Task Force Common Information Model (DMTF CIM):** provides a common definition of management information for systems, networks, applications and services, and allows for vendor extensions [65]. CIM's common definitions enable vendors to exchange semantically rich management information between systems throughout the network.

- **Web Services Distributed Management (WSDM):** manages resources and Web services. WSDM is defining two sets of specifications, which are [66]:

- **WSDM Management Using Web Services (MUWS):** defines how to represent and access the manageability interfaces of resources as Web services. It is the foundation of enabling management applications to be built using Web services and allows resources to be managed by many managers with one set of instrumentation. This specification provides interoperable, base manageability for monitoring and control managers using Web services.

- **WSDM Management Of Web Services (MOWS):** defines the manageability model for managing Web services as a resource and how to describe and access that manageability using MUWS.

- **Web Service Framework (WS-Framework):** is to define a generic and open framework for modelling and accessing statefull resources using Web services [67]. This includes mechanisms to describe views on the state, to support management of the state through properties associated with the Web service, and to describe how these mechanisms are extensible to groups of Web services.

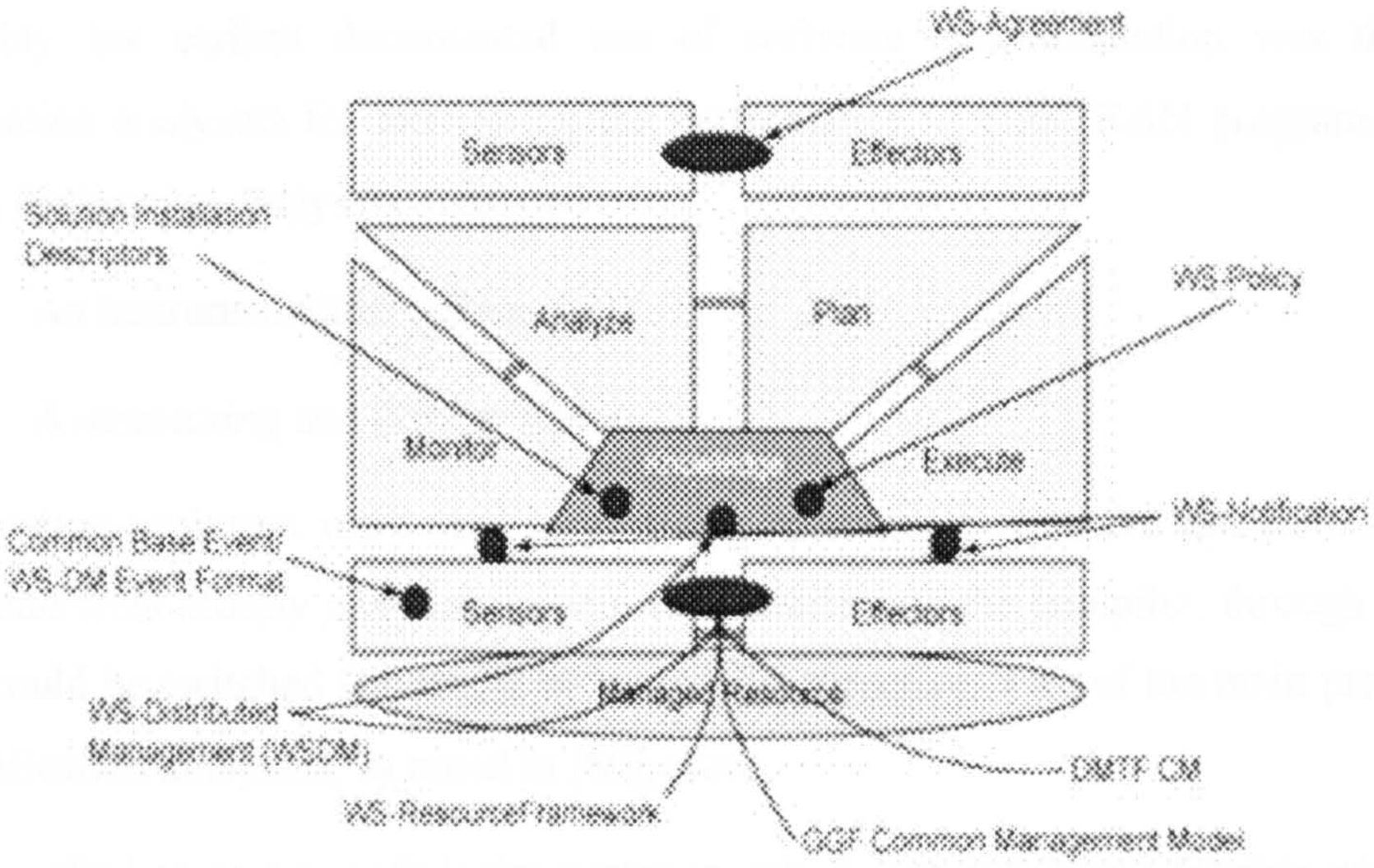


Figure 3.2: Autonomic Computing Interoperability Standards [42]

3.4. Monitoring and Self-Awareness

As indicated in Sections 3.1 and 3.2, autonomic capabilities depend on gathering systems state data on the self and the environment to determine the required course of action when required. Hence, in view of the vital importance of sensing and actuation the following section discusses monitoring systems and its constituting components namely software instrumentations (sensors), analysers and actuators.

3.4.1. Software Instrumentations

Software instrumentation is defined as the process of placing probes into software in order to record data [4]. Originally, instrumentation was employed to debug and test applications that run on single processor machines and for performance analysis of real-time systems. The parallel computing community later adopted instrumentation to debug, diagnose, evaluate and visualize parallel applications. More recently, distributed application developers have recognized the potentials of instrumentation for dynamic software analysis monitoring and management of distributed applications.

Probably the earliest documented use of software instrumentation was that of application analysers for monitoring the performance of FORTRAN programs [68]. These application analysers consisted of:

- An instrumentation system
- A monitoring and display system.

Application analysers were used as part of a more comprehensive test environment [69], and were closely associated (even integrated) with the compiler, through which they could be switched On or Off by a compiler directive. Two of the main problems of application analysers, as noted in [69], were:

- relied on source code instrumentation, which was not always feasible when the program relied on additional pre-compiled libraries.
- the instrumentation code often affected program performance, which presented problems in real-time applications.

Significant advances in the use of instrumentation came from the parallel computing community through the development of support tools for debugging, evaluating and visualizing parallel programs [70]. This community applied instrumentation in a much more structured manner, which followed a "tools" or "library" based approach [71]. Heath and Etheridge [72] listed the visualization tools, which were usually based on GUIs developed using graphics libraries such as OpenGL, Tcl/Tk and X/Motif that provided the user with a consistent view of the application and its environment. Performance, which is significant in parallel programs, was evaluated using unobtrusive instrumentation that did not carry an additional computational overhead. The parallel computing instrumentation tools were capable of synchronizing with applications and providing limited interaction facilities, but they were still generally static in nature. Static instrumentations are useful but not enough to record all changes of the environment. This fact is noticed by many developers and researchers who have recognised the vital of providing dynamic instrumentations to support the planetary-scale environment.

3.4.2. Intelligent Monitoring System

Most of the monitoring systems nowadays are "dumb" systems that depend on the monitoring system requester to provide the policies and rules for performing the job. While such monitoring system should sense and report changes in the environment in accordance with a given gauge (normal operation specifications), the generation of new or modified specifications of normal operation and/or monitoring policies are not supported. Therefore, research is required to address the evolution and intelligence of autonomic monitoring systems [46, 48, 51, 73].

3.5. Summary

In this chapter, the vital need for self-management system is presented in order to generate survivable systems that are able to handle unexpected behaviours and/or activities of the environment. Autonomic computing is anticipated to carry out automated and intelligent jobs that are required for performing self-management system. The standard model of autonomic computing has been defined which consists of four basic functions, which are: monitor, analyse, plan, and execute functions. The self-management systems and autonomic computing capabilities are adopted in many enterprise applications, next chapter present literature survey of these applications.

CHAPTER 4

LITERATURE REVIEW

4.1.Introduction

The development of the proposed self-management model presented in this thesis was inspired by previous works and based on a number of techniques emerging from distributed software engineering. Hence, as a convenient approach to presenting the landscape of previous and related work the review is structured into four main areas including:

- Planetary-scale enterprise applications: focusing on the use of grid computing to support large scale enterprise application.
- Autonomic computing: focusing on the work that addressing the autonomic computing capabilities.
- Self-management middleware: emphasising on the existing approaches and models for self-management middleware.
- Software instrumentation: concentrating on the existing monitoring resources for the Grid computing and PlanetLab environment.

The Chapter concludes with an outline of the main requirements for designing self-management viable distributed system.

4.2.Autonomic Grid Computing

Grid computing has been applied in a number of domains including;

- ***Health informatics;*** for instance to support not only computation intensive medical data mining and analysis, but also to enabled healthcare applications

to access and store clinical decision making data [74]. The most interesting benefit for the e-health applications is in the development of business processes provided by healthcare organizations to improve patient healthcare [74]. This is employing sensors technology for remote patients' health monitor. Where planetary-scale system (grid computing) provides a platform for storing and analysing high dimensional data such as cancer data [75]. Such projects are the e-Diamond project [75], InSiteOne (Digital Image Storing and Archive Service) [76] and 2nrich [77].

- ***Bioinformatics:*** typical bioinformatics databases can reach terrabytes in size, which lends itself to the application of high performance information storage, search and retrieval. A number of bioinformatics projects applying grid computing include; Folding [78], Protein Data Bank (PDB) [79] and Gene Expression Database (GXD) [80].

- ***Physics and astrophysics:*** these communities used grid computing to support computation intensive analysis and experiments. Moreover, such framework provides tools and services for education and outreach web site is primarily designed to promote learning into a scientific program of participating physics and astrologic. Such grid computing projects are GriPhyN [81], Large Hadron Collider (LHC) [82] , iVDGL [83], AstroGrid [84], and VISTA [85].

Evidently, the next maturity step up of most of the above described research-based grid applications will require high-availability, assurance, security guaranties, which should require autonomic and self-managing support for instance;

- to fault diagnosis and recovery
- To tune and adjust its performance
- To audit service usage in accordance to customers contract and policies
- To protect the applications from internal/external attacks.
- Autonomic computing vision has been embraced by many grid computing research and development communities. For instance, major IT players including; Sun Microsystems, Hewlett-Packard and IBM reported that the

approach is leading to improved high-availability, whilst reducing IT operation and administration costs [86].

4.2.1. Reference Models

As discussed in Section 3.3.3, the IBM blueprint is one of the widely known autonomic computing reference model, which is described by Fellenstein [5] as:

> *"...an overview of the basic strategic prospective, architectural concepts, technological constructs, and behaviours of building autonomic capabilities into on demand Operating Environment..."*

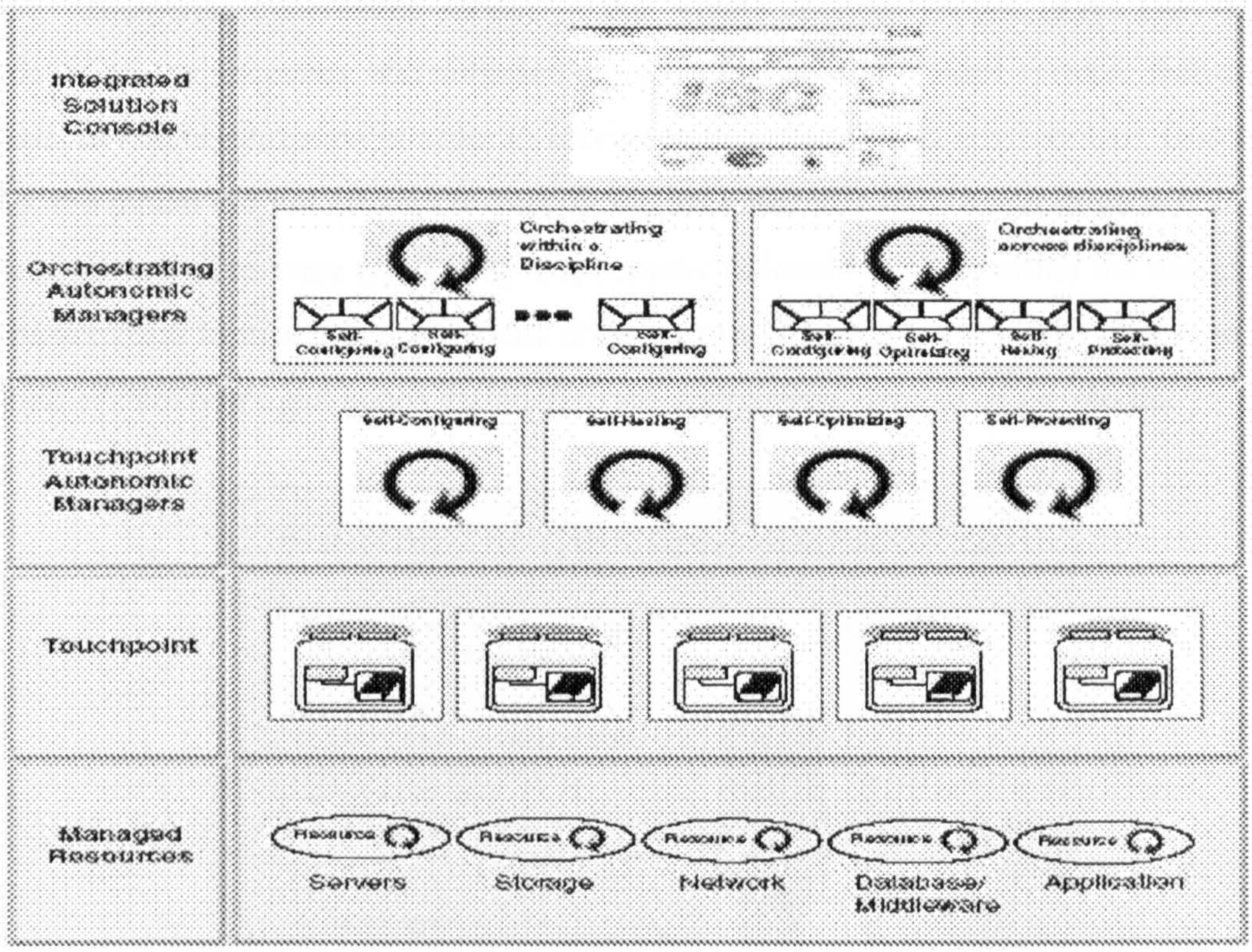


Figure 4.1: Autonomic Computing Vision [59]

The IBM blueprint has been promoted by the Company itself as a key instrument for understanding and supporting the requirements of the "*on demand business*" strategy [5], which will engender a new level of integration between the processes and applications inside the businesses' applications. Such integration will power e-business applications and environments (Fig. 4.1).

However, IBM blueprint requires further investigation to define a number of aspects including; (i) how to specify and operate policies-based autonomic systems' governance, and its interplay with systems deliberation (decision-making) [5, 15, 17]. (ii) How to insure interoperability and information exchange between the different actors of a given system while ensuring security and privacy. (iii) How to collecting, analysing and filtering readings, and enable interaction between autonomic computing components and its environment.

Many other research reference models have been proposed such as; Kinesthetics eXtreme (KX) Architecture [87] and J-Reference [88]. The latter was an extension of the managerial cybernetics model -- the Viable System Model [89]. The latter identified the necessary and sufficient communication and control systems that must exist in any organization in order to survive with a changing in environment. In doing so, the model did not attempt to specify the activities that must occur in each system. Instead, activities were typified by a cybernetic rationale to allow either the design of the activities to match the cybernetic criteria or for actual activities to be identified by their system types and hence assigned to the appropriate element of the model.

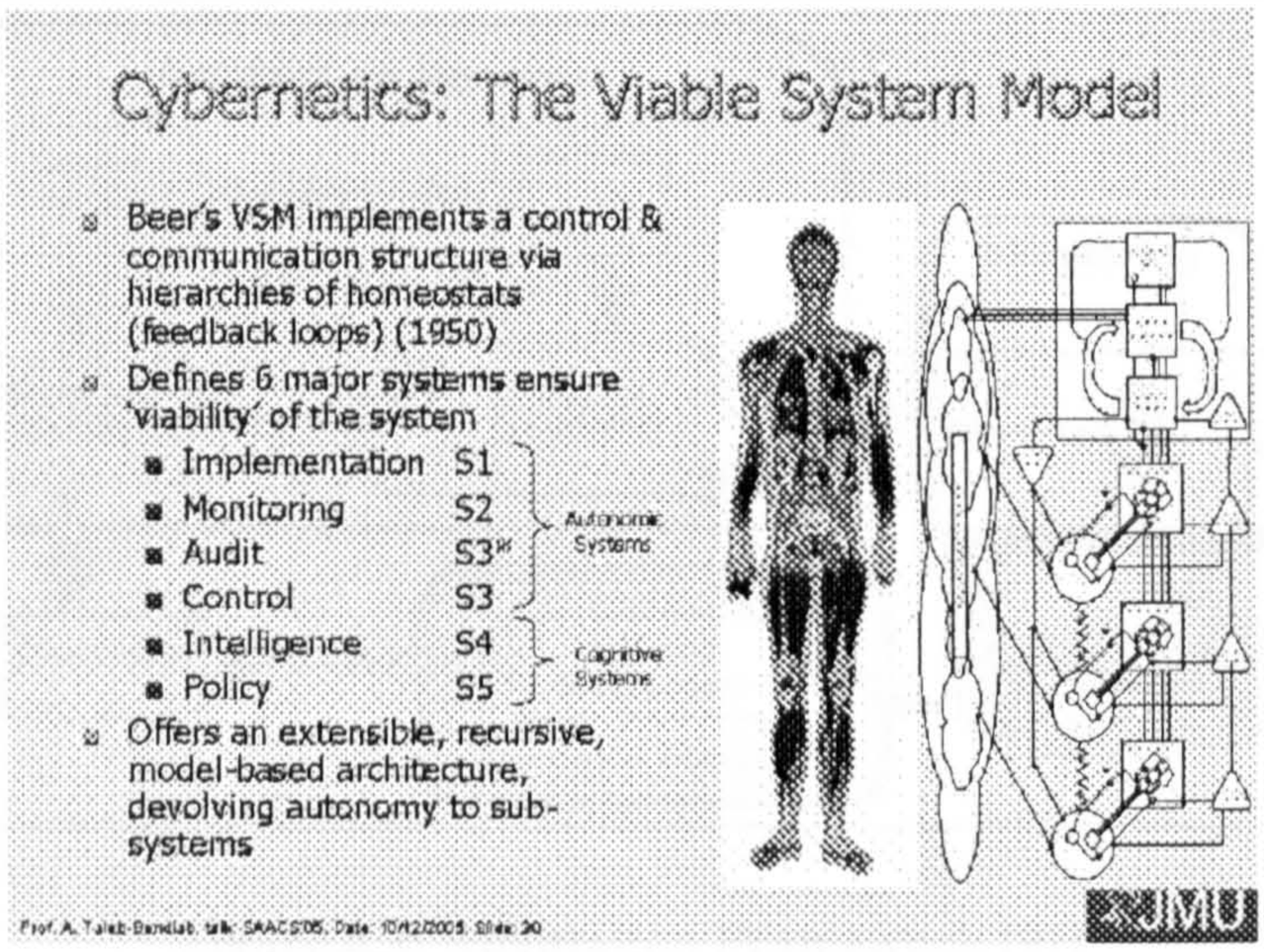


Figure 4.2: VSM Model [90]

The six major systems described by the VSM were (Fig. 4.2):

- System one (S1) – Operations

- System two (S2) – Coordination
- System three (S3) – Control
- System Three * (S3*) – Audit
- System four (S4) – Intelligence
- System five (S5) – Policy

Such a generalized approach allows the model to be applied to any organization regardless of size, structuring and guarantees monitoring, diagnosis and re-planning to adapt to change -- hence, this model as argued by Laws *et al.* [88, 90] provides a suitable design model for self-managing systems.

Many other research results proved key advancements in the understanding of design requirements of the different autonomic capabilities. For instance, Garlan *et al.* [91] developed another architectural model for self-healing systems, based on monitoring, problem detection and repair. The use of architectural models as the centrepiece of model-based adaptation had been explored by a number of other researchers [92].

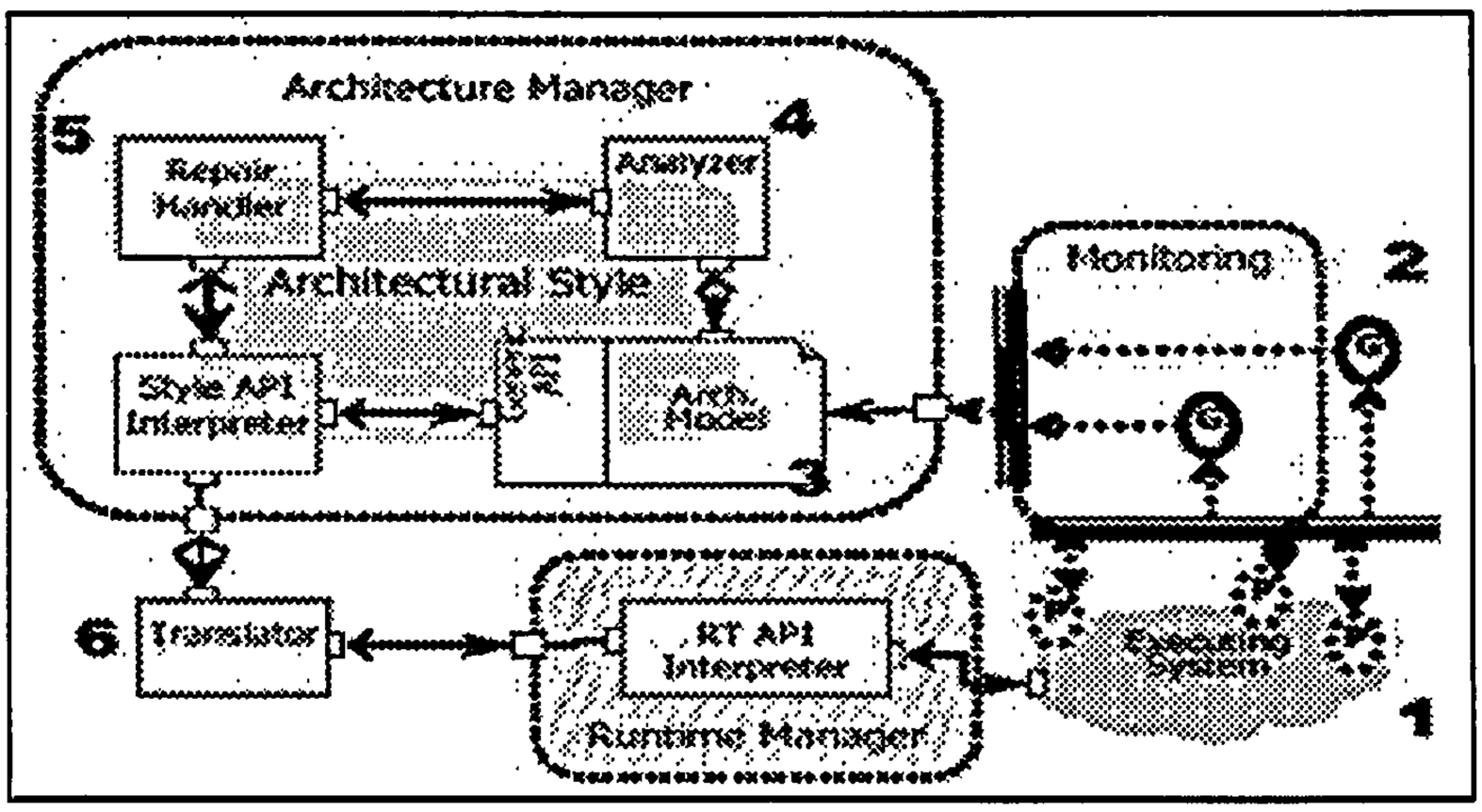


Figure 4.3: Adaptation Framework [91]

As illustrated in Figure 4.3, this architectural model provided support for runtime monitoring and software transformation to support self-healing. For instance, the architecture consists of an executing system (1), which is monitored to observe its run time behaviour (2). Monitored values are abstracted and related to architectural

properties of an architectural model (3). Changing properties of the architectural model trigger constraint evaluation (4) to determine whether the system is operating within an envelope of acceptable ranges. Violations of constraint are handled by a repair mechanism (5), which adapts the architecture. Architectural changes are propagated to the running system (6).

Badr [7] extended current models of self-adaptive software and reflective middleware with deliberative control mechanisms, which led to the proposal of a novel autonomic control middleware. This is to support the design and lifetime management of for deliberative middleware and application services. The developed approach was used as a reference model to facilitate a normative self-governance model that supports the safe self-adaptation of distributed applications for lifetime application management.

Pereira [93] used the VSM model to describe the fundamental requirements for a software framework and associated middleware services in order to develop on-demand application services based on employing self-healing capability. Moreover, she provided better understanding of software self-healing requirements for autonomic distributed software engineering, where she presented a reference model for self-healing capability.

4.2.2. Design Models

Bustard *et al.* [94] proposed an autonomic software design model based on an integration of Checkland's Soft Systems Methodology (SSM) [90] and Beer's Viable Systems Model (VSM) [95, 96]. The SSM offers a systematic and systemic structure with which to unravel complex situations using basic principles of system thinking. As shown in Figure 4.4, the model follows a top-down approach in a number of stages including;

- **Environment Design:** This is achieved using SSM method, which refined using the VSM model.
- **System Design:** which is achieved using a combination of methods including SSM and other computing-oriented modelling techniques such as; UML or

Other Design Technique (ODT). Again this design can be tuned by evaluating it with respect to VSM in the fourth and final stage.

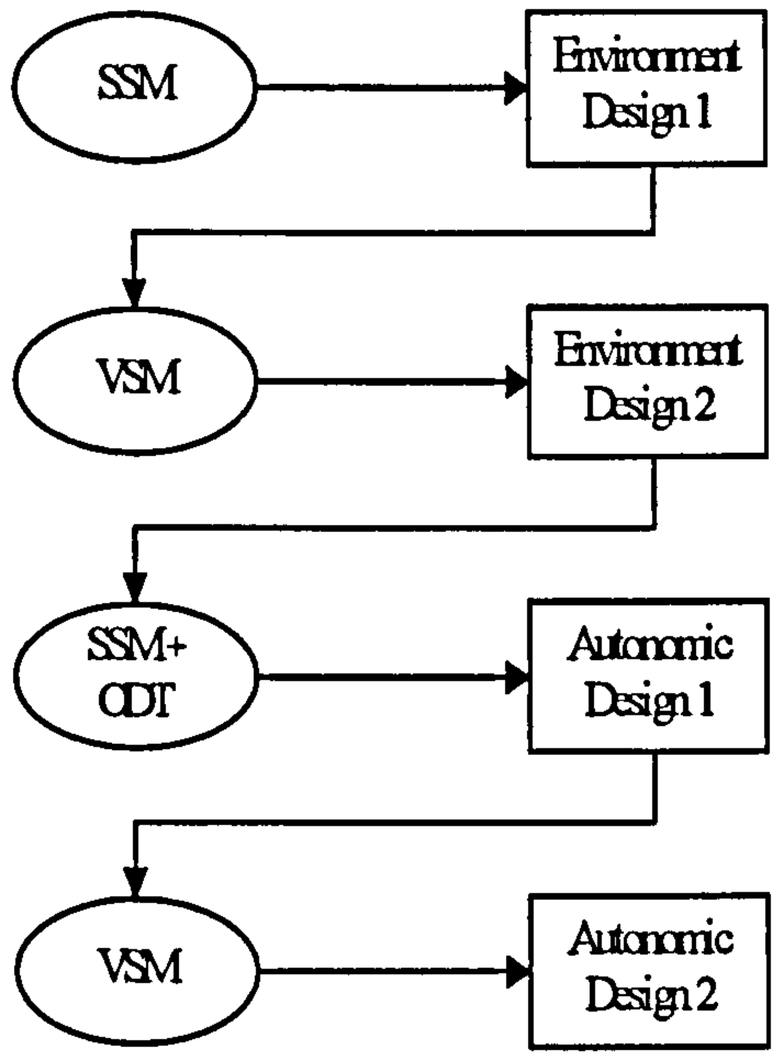


Figure 4.4 A Systems-Oriented Autonomic Design Process [90].

The proposed model makes reference to the need for appropriate design pattern, similar to Herring [97], which concentrated on the technical aspects of software viability. The latter is defined as the quality a software system has if its architecture can be adapted over time by humans (adaptable at design-time) toward becoming an "intelligent" control system (adaptive at runtime). The Viable System Model was described using Alexander's pattern language [98] form and related to software architecture. This pattern language showed how software may be evolved to be viable. As illustrated by Figure 4.5, the patterns can be outlined as follows [97]:

Separate Control: is the "redundancy" needed by the system to maintain stability within the environment.

Operational Control: caters to the immediate needs of the system.

Regulator Centre: maintains schedules of activities and provides for coordination of system activities with other related systems.

Sporadic Audit: verifies that all is as it should be relative to its directives.

Adaptive Control: focuses on the external aspects of the system. This function anticipates future states of the environment in which the system is embedded.

Supervisory Control: it constrains the possible actions of Operational and Adaptive Control to be consistent with overall, long term purposes.

Alerts: provides a mechanism for "lower" systems to communicate special conditions and emergencies.

Recursive Composition: is indicated in the inset to the right of the main figure.

Homeostatic Loop: is used to join all elements together.

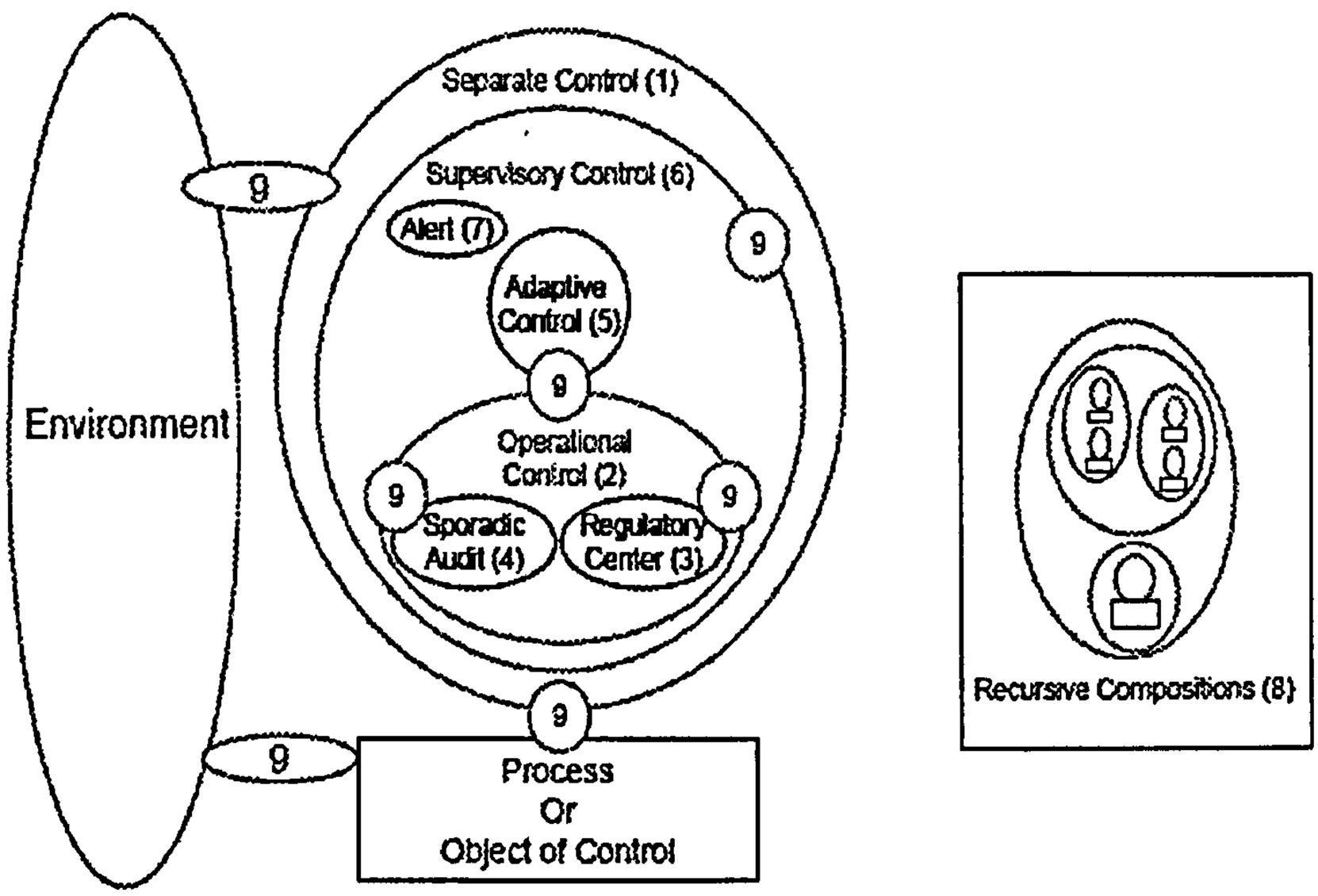


Figure 4.5: Relationships of the Patterns [97]

Herrings [97] and Pereira [93] provided a substantial insight into the requirements and exemplar software design patterns for some autonomic behaviour. However, it is difficult to assess their coverage and applicability to runtime adaptation issues of complex systems, that is, for widely distributed and heterogeneous systems.

4.2.3. Developing Autonomic Utilities

Much research is now underway to study the use of machine learning techniques to support the task of self-management, self-configuration, self-protecting, and other

general QoS improvement. For instance, M. Chen *et al.* [99] reported on their application of the C4.5 decision tree algorithm and data mining to categories causes of failure in large Internet sites such as eBay. Such failure had been recognized by many researchers. They showed the importance of using autonomic computing system for recovering such failure [93]. G. Candea, *et al.* [100] presented an Automatic Failure-Path Inference (AFPI) as an application-generic and automatic technique for dynamically discovering the failure dependency graphs of componentized Internet applications. They focused on applying AFPI to applications built on Java 2 Enterprise Edition middleware. AFPI-generated f-maps (fault-routes map) correctly omitted dependencies that appeared in the static call graph but did not result in observed fault propagation at runtime.

The accuracy of applying autonomic system using machine learning or data mining algorithms for large, distributed, and dynamic application environments is one of the critical problems. M. Chen *et al.* [1] presented a dynamic analysis methodology that automates problem determination in these environments by 1) coarse-grained tagging of numerous real client requests as they travel through the system and 2) using data mining techniques to correlate the believed failures and successes of these requests to determine which components are most likely to be at fault. They implemented Pinpoint, a framework for root cause analysis on the J2EE platform that requires no knowledge of the application components. In large scale system, there is an expectation for large number of failure services; this produces the demand for failure management system. M. Chen *et al.* [101] presented a new approach to manage failures and evolution in large, complex distributed systems using runtime paths. They used the paths that requests follow as they moved through the system as their core abstraction, and their "macro" approach focused on component interactions rather than the details of the components themselves.

Kumar *et al.* [102] presented a self-adaptation algorithm that has been designed to scale efficiently for thousands of streams and aims to maximize the overall business utility that attained from running middleware-based applications. They were focusing on the scalability and decentralised factors in designing their algorithm. From self-reconfigure middleware prospect, V. Kumar *et al.* [102] presented a self-adaptation

algorithm to increase the flexibility of the middleware to react for changing in the network conditions and business policies. They used hierarchical node partitioning scheme to deal with the scalability and decentralized system. But according to the lack of control unit in the existing infrastructure (Internet), this algorithm still has a gap in the real world of dealing with non-autonomous system.

4.3. Autonomic Middleware

Autonomic middleware can be characterised as a specialised middleware, which can provided autonomic capabilities[1] to distributed applications including legacy to exhibit self-managing behaviour [5].

4.3.1. Self-Management Middleware Services

Self-management middleware service is about shifting the managing burden of planetary-scale systems from people to technologies [59]. In addition, many researchers have highlighted the needs for self-management middleware to improve the systems' reaction to expected and unexpected changes in the planetary- scale systems. Blair *et al.* [103] described the use of self-management in the automatic (re)configuration of overlay networks, which was achieved using reflective middleware services. Others, Babaoglu *et al.* [104] described a proposed load-balance protocol for self-managing distributed systems. This was illustrated via a simple experiment that consists of injecting components (like monitoring, searching, clustering, control and sorting) that trigger and test the load-balancing protocol. As illustrated in Figure 4.6, and concurring with results presented in Kephart *et al.* [51], Wail *et al.* [15, 17], this work identified a set of mechanisms necessary for self-management including; control loops, resource management and measure, and analysis and decision making units.

[1] Capabilities such as self-healing, management, etc…..

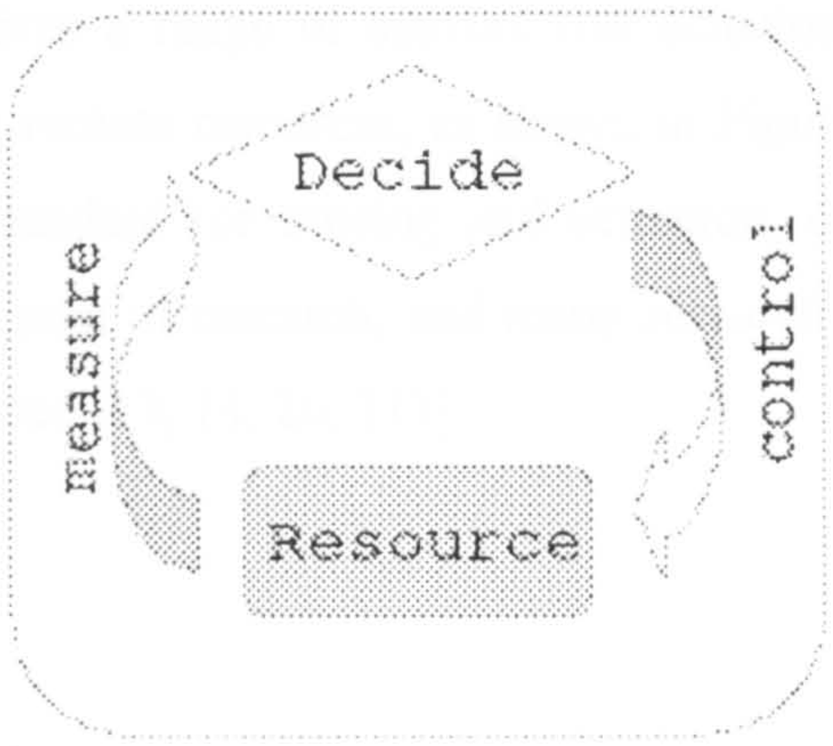


Figure 4.6: Self-Management System Model [51]

4.3.2. Management of Web Services

Recent research highlighted the need for interoperation standards and reference models to support the management of decentralised web services and remote resources [12]. A number of interoperation models already existed in the public domain including; integration middle layer architecture [105], data interchange standards [106, 107] providing via serviceware which is an integration layer between systems' managements and managed resources. Hence, Web Services Distributed Management (WSDM) had been proposed and developed to provide a standard for distributed web services management [108]. Based on web services protocols and standards, WSDM enables for instance: the remote resources management and control. As described by [109, 110], WSDM consists of two specifications, namely;

- **WSDM Management Using Web Services (MUWS);** which defines how to represent and access the manageability interfaces of resources as Web services. Moreover, it provides a standard management event format to improve interoperability and correlation.
- **WSDM Management of Web Services (MOWS),** which defines how to manage Web services as resources and how to describe and access that manageability based on using MOWS. In addition, MOWS provides mechanisms and methodologies that enable manageable Web services applications to interoperate across enterprise and organizational boundaries.

Arguably, WSDM requires a range of sensors and actuators in order to auditing, analysing and managing remote resources, as shown in Figure 4.7. Though, WSDM does not define yet a standard for sensing and actuation, or a related framework. Though, this is a fertile area of research, and many research frameworks for sensor and actuator already existed [13, 14, 20, 111].

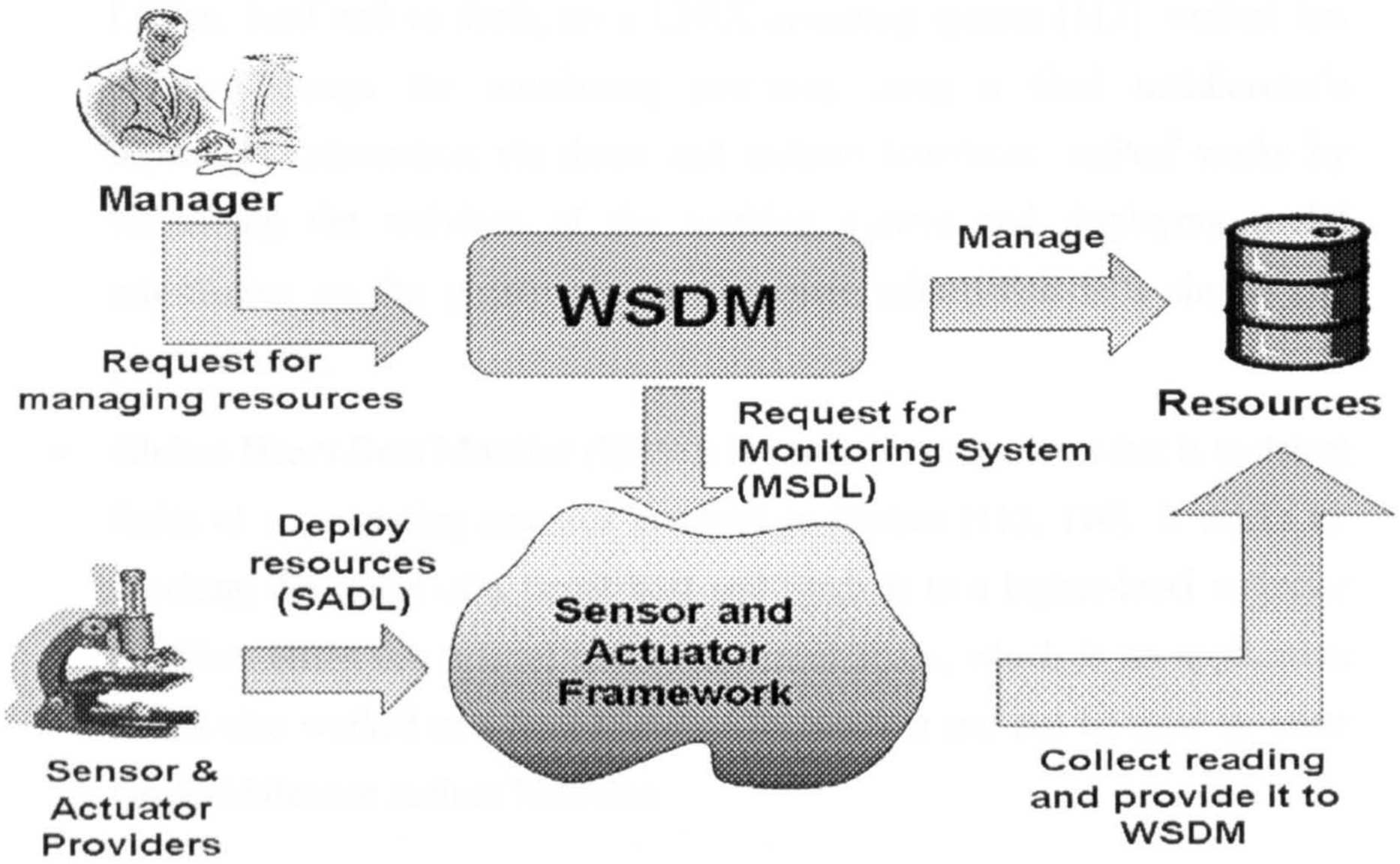


Figure 4.7: Sensor and Actuator Framework for WSDM

4.3.3. Software Instrumentation for Planetary-Scale System

Much work related to systems monitoring for grid computing is now widely published [20, 112], describing numerous monitoring models including heart beat monitoring, on-fly monitoring, and other type of dynamic monitoring model . Reilly and Taleb-Bendiab [113] described a dynamic instrumentation framework, which provided support to monitor and manage Jini [114] applications based on-the-fly monitoring model. The framework adopted a service-oriented programming model and the software factory pattern to dynamically generate specific instrument types, which were deployed and interfaced to client services via Java's dynamic proxy API and Jini's remote event. This enabled on-demand insertion and removal of instrumentation services. Other types of monitoring system are presented in the following types:

- **visPerf:** is a type of monitoring system for Grid computing in which several computing entities are used to solve a computational problem with parallel processing and distributed computing components. Initially, this work was executed as a simple monitoring capability for a specific system, NetSolve [115]. But, it can be extended to provide a monitor for Globus, Condor, Legion, Ninf and so forth, on a UNIX operating system [112]. visPerf has different design for monitoring activities using a Grid middleware's dependent information via direct and indirect interfaces. visPerf works by visualizing the activities of the working system and displaying useful information on the system plus performance information in a simple and practical way.

- **Globus Heart Beat Monitor (HBM):** is a monitor component that is to detect faults of a computing resource involved in Globus [112, 116]. It works by checking the status of a target host and sends it to a higher-level collector machine. HBM has a level checkpoint mechanism, which is an application that is also worked as a fault-tolerance mechanism and can be used by other Grid middleware such as NetSolve.

- **GridMonitor Java Applet:** is a kind of monitoring for Globus system [112, 116]. It shows the Grid information and server activities and status for all sites which registered with *Globus Metacomputing Directory Service* (MDS) and *Java Agents for Monitoring and Management* (JAMM) [117]. JAMM is an agent-based monitoring system for Grid environments that makes the automation and execution of monitoring sensors possible and it works to collect the event data. It supports the system's activities and general performance including network traffic and hardware resources usage like CPU. An alert service is also embedded to this type of monitoring in order to notify the thinking system in case of overloaded.

- **Grid Monitoring Architecture (GMA):** is employed to monitor the resources usages over the grid, such as: memory, network and applications usages [4]. The collected data is converted by the "*producer*" to data events.

GMA is supporting both a subscription model and a request/response model. The unique feature of GMA is that performance monitoring data travel from the producers of the data directly to the consumers. The GMA architecture specification ignores many details that are necessary to build interoperable monitoring systems. Several researches and projects groups are now developing monitoring services depending on this technique and architecture, such as R-GMA (relational OMA, and it called so because it uses a relational model for all data and uses tables to organize the Grid entities data), Remos, and TOPOMON [4].

- **Windows Management Instrumentation (WMI):** that consist of three parts which are [111]:
    - *Management Infrastructure:* There is an object manager, called Common Information Model (CIM) which is used by the consumers in order to do the communication process between management applications and providers.
    - *Managed Objects:* Management applications get access to manage objects using the CIM Object Manager.
    - *WMI Providers:* WMI providers are components, which supply dynamic management data with the required information regarding managed objects, handle specific object requests, or fire WMI events.
- **Application Center (AC):** is designed specifically for e-site management, which cover the Web server and Web farm applications, in addition to Web Services deployment [111].
- **Microsoft Operations Manager (MOM):** is further more of a common purpose management and monitoring tool. It is used to monitor BizTalk Server environments, or SQL Server [111].
- **Enterprise Instrumentation Framework (EIF):** is a framework technology for monitoring and troubleshooting high-volume systems and distributed environments for Microsoft Windows environment [111]. EIF works with

Application Centre (AC) and Microsoft Operations Manager (MOM) hand-by-hand, providing a standard data for event management, tracing and logs.

4.3.4. PlanetLab Software Instrumentation

PlanetLab environment is designed and developed in order to offer testbed tools for carrying out research on web development [38, 118, 119]. Therefore, much work focused on systems monitoring including; tools and services for PlanetLab in order to offer a fabric for follow-up the developed services. Different types of sensors' services and API such as; CoMon [120, 121], Ganglia [39], iPerf [122] and IrisLog [123] are available for this environment. These are used to provide instrumentation data from the resources such as; processor, process, memory, transmission rate, bandwidth and other networking services. Other tools, such as; Node List [124], Trumpet [125], and SWORD [126] are used to provide the availability guarantied and status information of available nodes (hosts) that registered with the PlanetLab environment.

Chun *et al.* [37] described the initial implementation of PlanetLab, including the mechanism used to implement virtualization, and the collection of core services used to manage PlanetLab. While, Matthew *et al.* [127] presented the design, implementation, and evaluation of Ganglia, a scalable distributed monitoring system for high-performance computing systems. Chen *et al.* [128] tried to employ these resources for presenting an algebraic approach for adaptive scalable overlay network monitoring. On the other hand, other works focused on sensors discovery mechanisms to support fault-tolerance of heterogeneous distributed systems. For instance, Karuppiah *et al.* [129] discussed the design of a distributed vision system that enables several heterogeneous sensors with different processing rates to exchange information in a timely manner in order to support the tracking of multiple human subjects and mobile robots in an indoor smart environment.

4.4.Summary

This chapter reviewed the current and related research to self-managing global computing systems including; research relevant to reference models, required infrastructures and middleware services, and management requirements.

Whilst, much progress has been made towards improved understanding of self-management system, though much work is still required to support large-scale (planetary-scale) self-managing systems [31]. For instance, to understanding design and/or manage planetary-scale system.

This chapter also investigated variety of approaches which are used to implement autonomic computing capabilities. This led to the definition of generic requirements for self-management, which will be described in details in Chapter 5.

CHAPTER 5

SELF-MANAGEMENT REQUIREMENTS

5.1.Introduction

A prevailing design model of autonomic computing systems is one of a goal-oriented and model-based architecture. Wherein, rules elicited from domain expert through traditional knowledge engineering techniques and/or domain analysis are embedded in meta-systems to provide self-management behaviour including [130]; self-protective, self-optimization, self-tuning, self-configuration, self-governance and/or self-healing. Such a rule-based (or policy-based [1, 54]) management approach as reported by many [5, 54, 55, 93] is appropriate for systems' self-management with inherently stable operating rules (and/or policies).

This chapter describes the general requirements for designing self-management system. This will be followed by a description of different autonomic computing concepts, as well as, tools, services and frameworks which are required to perform the expected job from self-management middleware system.

5.2.Self-Management Model Requirements

Self-management middleware service is here proposed to imbue heterogeneous distributed software applications with autonomic capabilities. Such a self-management middleware service requires the integration and access to a range of utilities, metamodels and frameworks, each of which are outlined and detailed below.

5.2.1.Description Languages

Different types of description languages are required to facilitate open standard interoperability and information exchanging between the different components and

users of an open large-scale and widely distributed system. These types of description languages are used to exchange information between:

- Different types of middleware
- Consumers and middleware
- Resources provider and middleware resources container
- Middleware and resources
- Middleware services and auditing systems

5.2.2. Frameworks

In this work a set of software frameworks are required to extend the core middleware services with self-management application services including; dynamic application services assembly, activation and deployment, and sensing and actuation for self-awareness and introspection. These frameworks can be summarised as:

- **Assembly Services and Infrastructure Framework (ASIF):** provides the fabric for deploying, discovering, invocating and managing services and infrastructures. These resources amalgamate together in order to perform the required tasks for the consumers taking in consideration the high availability, fidelity, QoS and reliability of the resources along with reducing cost of ownership.
- **Sensor and Actuator Framework (SAF):** provides a layer for monitoring and controlling planetary-scale system by offering variety types of monitoring resources from sensors, actuators and analysers services which are provided from different vendors. Moreover, such framework provides standard logger system, which in its turn represents a fabric for collecting readings in semantic format in order to be delivered to the consumers.

One of the main objectives of the suggested frameworks is to improve the fidelity of discovering resources according to the searching parameters. Sensor and actuator fidelity, robustness and assurance are some of the major concerns considered by the proposed and developed sensor framework. Such concept are borrowed from those

developed by the intelligent systems engineering community including; self-convergence, self-optimisation, self healing and self-adaptive.

Many critical concerns in the sensors and actuators overlay are the on-demand selection and access to the correct monitoring resources type for a given monitoring task. Thus, the SAF is designed in order to assist the consumers in nominating the most appropriate type of resources according to their requirements. The same idea is also applied for ASIF, which is to assist the consumers to find the most appropriate services or infrastructures according to their needs.

At this stage, an integration of autonomic system as a service with the ASIF and SAF is proposed to generate and develop an intelligent frameworks. These frameworks are able to augment its QoS and response in electing the demanded resources. Such intelligent services require a wealth of information in order to carry out the intelligent matching between consumer request and available sensors take in the account the nature of the application. Therefore, three types of description languages are proposed developed and implemented to provide ubiquitous, interoperability and formalised access to resources metamodel in order to assist in the discovery and selection of required types of resources. These languages are *Assembly Services and Description Language* (ASIDL), *Sensor and Actuator Description Language* (SADL) and *Monitor Session and Description Language* (MSDL).

5.2.3. Services and Utilities

Several services and tools are necessitated for the self-management service to carry out the expected jobs from the automated system as well be demonstrated in Chapters 6, 7 and 9. Such services are supposed to be one of the core functions of the middleware to ensure the inheritances of the control policy from the middleware itself. Some of these services and tools are listed below and shown in Figure 5.1:

- **Dynamic Sensors:** to gather information from the decentralised, heterogeneous and distributed environments based on generating on-fly sensors.

- **Logger System:** to store the collected readings in profile inside the distributed logger system. Moreover, this unit is in charge of delivering the collected readings to the consumers in open standard format, such as XML.
- **Analysis Services:** to perform the analysis tasks for the collected readings those are obtained from the monitoring process. Such services adjust/tune the environment, service, infrastructures and/or middleware to achieve the most fitted model for each application according to its nature.
- **Service Level of Agreement (SLA):** to manage the contract between different parties of the system like management services, frameworks, sensors, consumers and others in order to insure the security level of the system and define the privilege of each parties.
- **Intelligence Services:** to execute the automated self-management tasks. Such services can be extended to carry out the management tasks for different types of distributed enterprise applications like connected-home machine and e-health systems. Therefore, such management services should be defined in semantic way to encourage other applications for adopting such intelligent services in performing the prediction or clustering processes. The result of the intelligent services is represented as decisions that are forward to the tuning and adjustment services.

The core of the autonomic computing service is the methods, algorithms and techniques which are responsible for looking after analysing and deciding the actions those are required to be taken from the autonomic services according to the characteristics of the environment changes. Mathematic analysis, machine learning technique and data mining methods are evaluated and measured over many years for doing variety of reasoning tasks for different applications such as those in business and market analysis [5], text, image and speech recognition [131], system management [5], fault-tolerance [129] and other types of applications.

The expected role of the intelligent service in the self-management scenarios is to leverage the deployment and use of the autonomic computing with different types of applications leading to the goal of self-management system. Machine learning and

mathematic analysis are tested in order to be used in heterogeneous, distributed and decentralized system over planetary scale system. These algorithms analyse and decide the way of monitoring, controlling, managing and adjusting environments' resources to achieve the goal of high availability, security, optimising, QoS with increasing the threshold of the system failure. Such services should be designed, developed, and implemented based on the algorithm of web service technology to accomplish the idea of open standard system which makes them compatible with any platform taken in considering the SLA between the services and the system.

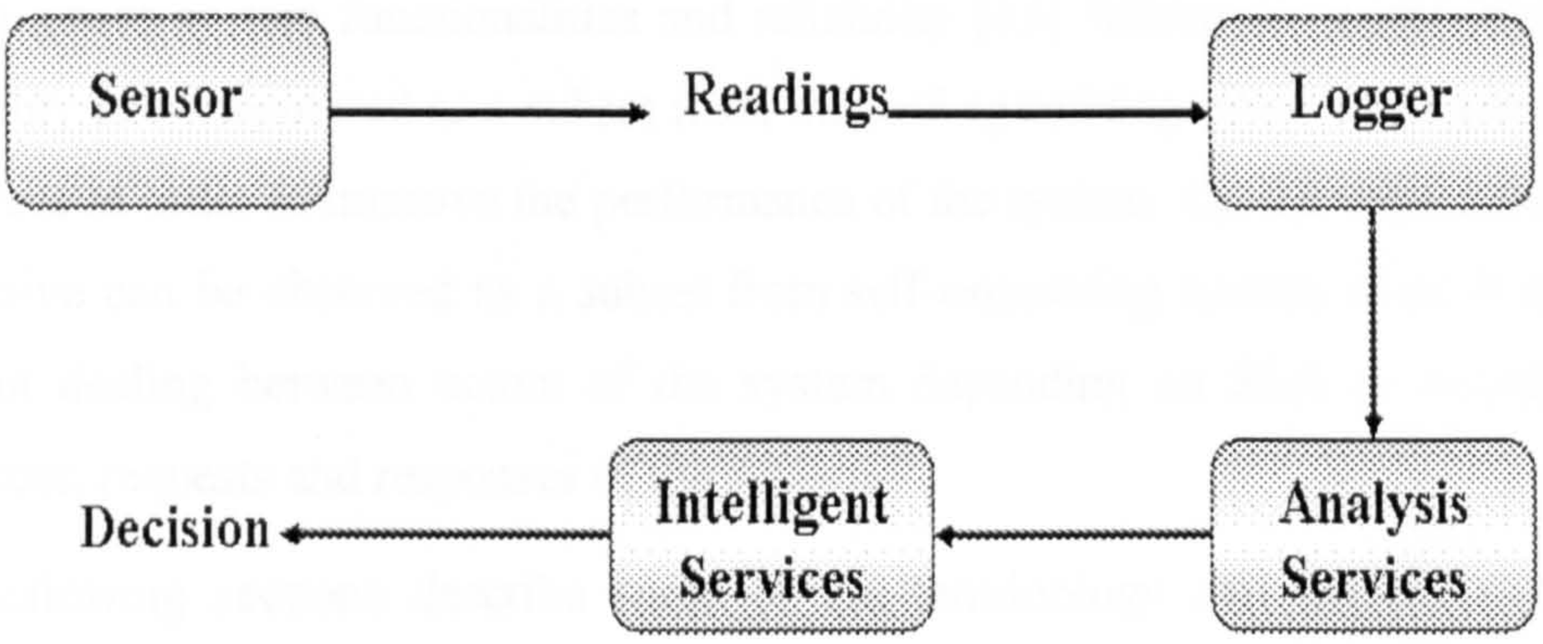


Figure 5.1: Services and Tools of the Self-Management System

Self-management middleware services require an effective monitoring and auditing utilities to facilitate applications' tuning, reconfiguration and adjusting. This auditing system requires an on-demand autonomic monitoring system to get information from the running environments and make scrutiny for the readings. Different types of sensors are required to cover all the demanded readings from different target of the grid environment. These monitoring resources are expected to be offered and deployed by many providers. Such resources are used by many consumers and monitoring system.

The monitoring resources should be managed in a way that guaranties the high reliability, fidelity, availability and security. Therefore, SAF is proposed through this work to support sensor and actuator generation, deployment, discovery and general management in order to achieve the above-mentioned goals.

5.3. Self-Management Middleware for Planetary-Scale Systems

This work proposes to provide autonomic behaviour via the use of self-management middleware services. These autonomic computing concepts and/or capabilities are presented by self-adaptive, self-configuring, self-healing, self-optimizing, self-organising, self-governance and self-protective. For example, self-optimising concept can be viewed as a subset from the self-adaptive since it provides a method to maintain a required level of services performance stand-in just on system resources with improve system functionalities and reliability [43]. Moreover, a self-organising capability can considered as a subset from the self-optimising since it reorganises the resources in order to improve the performance of the system. On the other hand, self-protective can be observed as a subset from self-organising system since it offers a base of dealing between actors of the system depending on SLA to organise the resources, requests and responses of the system.

The following sections describe some of the terminology and taxonomy for the autonomic computing concepts and capabilities for the planetary-scale system.

5.3.1. Self-Organising

In the self-organising system, the resources of the planetary-scale system and consumers are beneficial from the autonomic computing services by executing variety of tasks, which can be summarised as:

- **Services & Infrastructures Management:** The self-organising capability is used in this case to arrange and mange the services and infrastructures fabric in the planetary-scale model in order to attain the high QoS, reliability, fidelity, availability of the resources along with least response time. To achieve these goals, autonomic computing is employed for organising the deployment of new resources and reorganisation the deployed resources.
- **Consumers' requests management:** The autonomic computing services represented by self-organising capability are suggested to be used in managing consumer's requests to offer the high accuracy of system's responses and

reducing the cost of ownership. The autonomic computing service should taking in consideration the priority and SLA between the consumers and the system. This will asset in developing self-protective system which is one of the autonomic computing capabilities.

5.3.2. Self-Configuration

Self-configuration autonomic computing services are serviceable for the services and/or infrastructures deployment and discovery processes to fulfil the concepts of the new enterprise business mental picture of the high availability, QoS and security in addition to the reduction of the cost of possession. Managing the discovery and deployment processes in automated way according to environments' policies and characteristics augment the positive reactions of the system's components against different dilemmas, which may have an effect on the performance of the system. Intelligent classification of the services and infrastructures along with other components of the middleware increase the response and fidelity of the system in identifying consumers' demands.

Moreover, self-configuration capability is responsible for the configuration of the deployment of the new components in order to insure the integration or replacement of the new components with the old components of the system. The scenario starts by deploying or publishing resources (services and/or infrastructures) with the middleware, as shown in Figure 5.2. The deployed resources can replace or merge the existing components in case the autonomic service discovers that the new configuration offers better service than the old one. All the resources are stored in the resources container (ASIF) to be available for all consumers. On the other hand, the consumers' requests pass through the autonomic services to the resources container. In turn, the autonomic computing service insures higher accuracy and better QoS depending on the use of the intelligent discovering algorithms inside the autonomic services such as: ranking algorithm, link state, indexing and others.

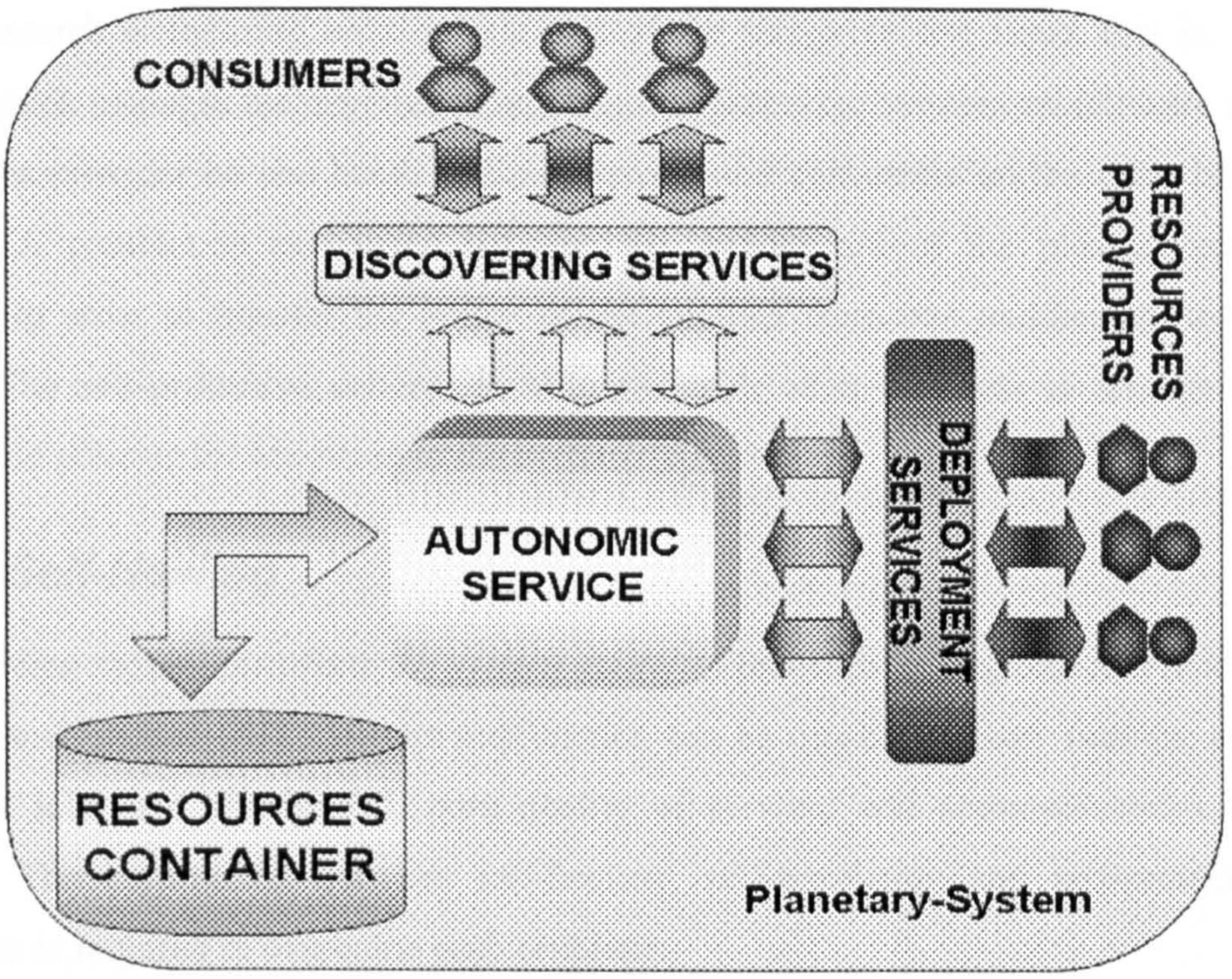


Figure 5.2: Self-Configuration Scenario

5.3.3. Self-Optimising

Planetary-scale system requires a range of management processes and services for making the middleware or broker interact with the services/infrastructures and applications faster. Self-optimising capability based on auditing system is an attractive solution to re-manage and re-adjust the system to offer better response in automated way. SAF is utilised to gather information from the environment in order to assist the self-optimising capability to execute the required tasks. In this work, self-optimising capability is utilised to perform the On-Demand Services (ODS), as will be shown in Chapter 9.

5.3.4. Self-Protective

Self-protective issue focuses on two dimensions. The first dimension is the security of the system and the overcoming of the attack from outside and/or inside. The second dimension is the negotiation and contract processes between the system and attached actors to define the privilege of the actors according to SLA.

This research is focusing only on one part of the self-protective issue, that is, the SLA, which manages the negotiation and contract processes between middleware and resources provider on one hand, and middleware and consumers on the other hand. The SLA contract is exchanged between the middleware of the planetary-scale system in order to transfer the policy of the contract from one middleware to the other. The self-protective concept is affiliated with other concepts to arrange the rights and policies for accessing to the systems' resources.

5.3.5. Self-Healing

Self-healing system manifests the system ability to investigate, diagnose and react to system malfunction [43]. System malfunction can be caused by many reasons, such as fault tolerance, overloaded, pure quality of services and communications and others. The corporation between self-healing, self-organising and self-protective concepts is required to overcome such problems.

Anticipating the load in advance increases the ability of the system to plan for in advance recovery of the resources (services and/or infrastructures) depending on different algorithm, like load balance, replication, mirroring and others. Autonomic computing service in this system is responsible for carrying out load prediction depending on the information available from the monitoring system taken in consideration the SLA of the consumers and services. This job is essential for the fault tolerance services to manage and recover services prior to the failure of the resources.

5.4. Design and Implementation Requirements

In line with the reference models (Sec. 4.2.1) a set of design requirements are identified as necessary for the development of the self-management system for the planetary-scale system based on the use of autonomic computing capabilities. These requirements are detailed below and summarised in Table 5.1.These requirements are addressed in the following points:

Self-Management Planetary-Scale Model: to design a model suitable for sharing and using resources in global environment that able to manage itself based on using

middleware layer (R-1). This model is proposed to support the zones/clouds model, which will assist in improving the managing, auditing and supporting processes. In addition, this generic model should be able to survive with the unexpected changes and behaviour in the distributed environment. Therefore VSM is suggested to be integrated with the self-managing planetary scale model to offer a survival planetary-scale model (R-2). This survival model will be integrated with a software model, like Gang of Four (GOF) [132-134], to make it acceptable in the software community and define the way of generating services (R-3).

Autonomic Computing Capabilities: to carry out the management process in automated and intelligent way for the self-management service (R-4). Different intelligent services are required by the autonomic computing to perform the smart and automated tasks of predicting and classifying processes which help in taking descisions (R-5). These intelligent services are built based on machine learning and statistical algorithm and should be as web services to be acceptable in OGSA.

Resources Framework: to generate a common container for deploying, discovering and invoking resources (services and infrastructures) (R-6). This framework is proposed to enhance the performance, fidelity and robustness of the resources.

Resources Description Language: to find common and semantic language between resources providers and consumers (users, applications or control systems) (R-7).

Monitoring System Model: to design, develop, and implement an auditing model for the planetary-scale system (R-8). Such auditing system will be utilised by the autonomic computing to track the changes in the behaviour of the system.

Monitoring Resources Framework: to create a common container for deploying, discovering and invoking monitoring resources (sensors, actuators and analysers) (R-9). Such monitoring resources are used by the monitoring system to collect readings from the monitored targets. This framework is proposed to enhance the performance, fidelity and robustness of the monitoring resources.

Monitoring Resources Description Languages: to describe the monitoring resources in semantic and open standard format. These description languages are

divided into two types, first to define the deployed monitoring resources (R-10) and the second to define the consumer's request (R-11).

Table 5.1 summarise the requirements for survival self-management service for planetary-scale system.

Table 5. 1: Requirements for Survival Self-Management Service for Planetary-Scale System

#	Requirements	Ref
1	Design model for self-management planetary-scale system that able to manage itself.	R-1
2	Design model for survival self-management planetary-scale system.	R-2
3	Design software model for generating services for the planetary scale system.	R-3
4	Designing, developing and implementing autonomic computing capabilities to support self-management services.	R-4
5	Designing, developing and implementing intelligent services to support the autonomic computing capabilities.	R-5
6	Resources framework for deploying, discovering and invoking different types of services and infrastructures.	R-6
7	Resources description language to define the resources in semantic and open standard format.	R-7
8	Monitoring system model to generate an auditing system for the global computing.	R-8
9	Monitoring resources framework to deploy, discover and invoke monitoring resources.	R-9
10	Monitor resources description language to define the monitoring resources in semantic and open standard format.	R-10
11	Monitoring requesting description language to define the monitoring resources requests by the consumers in semantic and open standard format.	R-11

5.5. Summary

The requirements for building self-management system for planetary scale system are discussed in this chapter. Reference models, autonomic computing, description languages, services and frameworks are considered as the vital requirements for designing, developing and implementing self-management model.

Two types of frameworks are addressed in this chapter to be the fabric for deploying, discovering and invoking resources. These two types are Assembly Services and

Infrastructures Framework (ASIF) and Sensor and Actuator Framework (SAF). Moreover, intelligent service is outlined in this chapter, where it is in charge of doing intelligent process like prediction, analysing and classification.

To this end, resources which are required to build self-management system for Global computing are described in order to prepare the material for designing, development and implemented the model. Next chapter describes the model of planetary-scale system based on existing Internet model. This model is utilised to build another model for self-management that able to survive in planetary-scale system.

CHAPTER 6

MODELLING & DESIGN

6.1.Introduction

To support the design and management of complexity widely-distributed grid systems, this work follows the hierarchical structuring, separation of concerns and self-similarity principles inherent in the Viable System Model (VSM) [95, 96]. The latter has been argued by Laws *et al.* [88, 90] to provide an expressive blueprint for autonomic computing (Sec. 4.2.2).

This chapter details a proposed Self-Management Viable System Model (SM-VSM) for global computing, which is based on an extension of the VSM design pattern and the Gang of Four (GoF) software design pattern. SM-VSM provides a reference model and a generative support for the design of self-managing planetary-scale applications. An illustrative example of modelling a self-tuning (self-optimising) system is here used to detail the proposed SM-VSM design pattern (Sec. 6.4).

6.2.Planetary-Scale Architecture and Middleware

In view of the heterogeneity of decentralised global computing, middleware is used to facilitating interoperability via distributed object-oriented programming. Such that planetary-scale systems are structured and organised into zones/clouds; where each of which is a federation of software services, and contain at least one cloud agent responsible for the following roles:

- Gateway: facilitating the information interchange with other clouds.
- Controller: managing the applications behaviour and this includes tuning the system to insure the QoS and SLA.

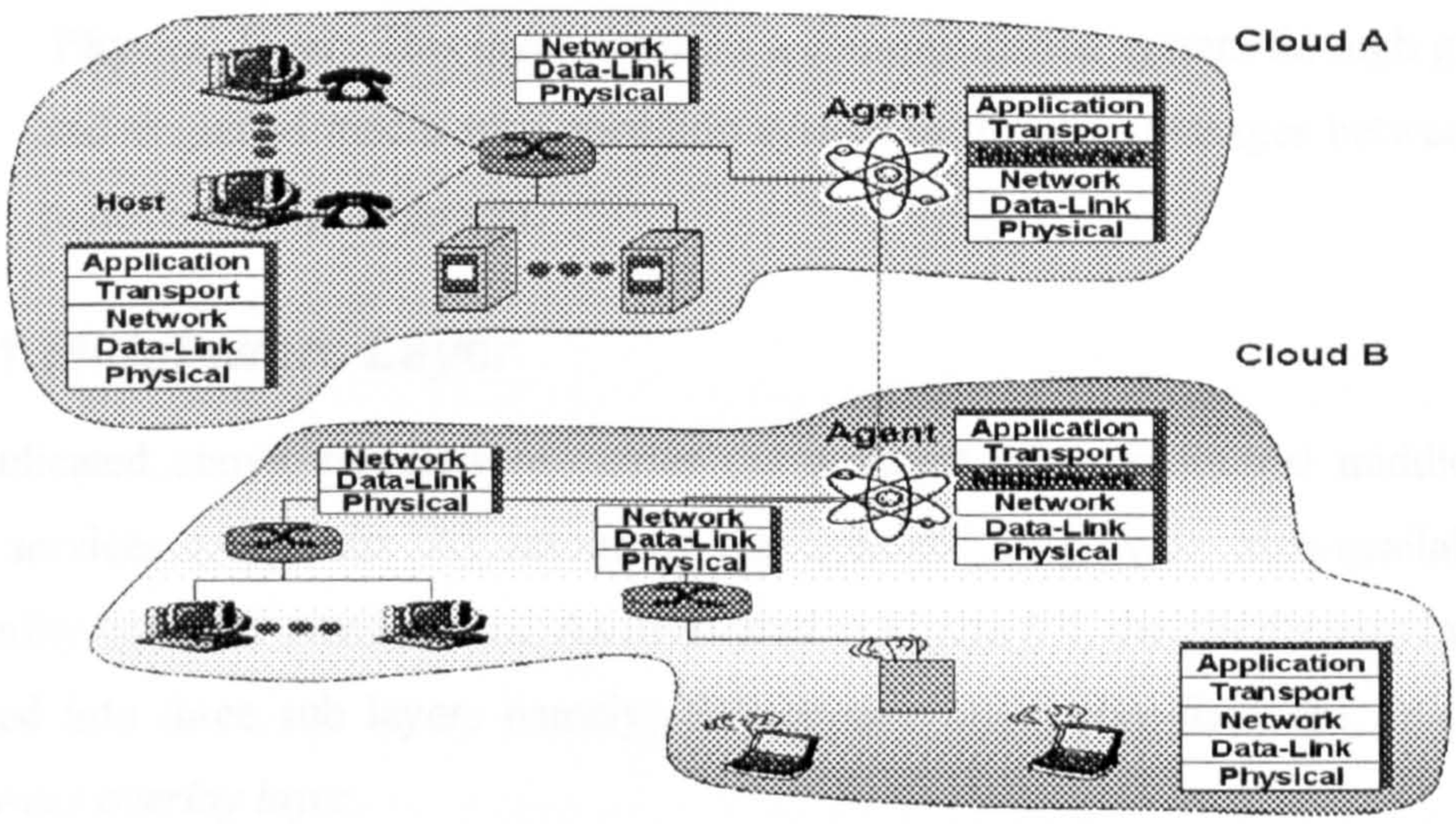


Figure 6.1: Layers of the Planetary-Scale System

As shown in Figure 6.1, the extended OSI layering for planetary-scale system can be summarised as follows [31]:

- **Application layer:** This layer is playing as a bridge between the user and planetary-scale model. This layer consists of user interfaces and user agents which run on the client side. This layer is responsible of generating the request and response messages between the open system communities.
- **Transport layer:** This layer is responsible for end-to-end delivery of the messages. Depending on the type of planetary-scale applications, different transport layer protocols are used; like TCP, UDP, and SOAP to provide variety of ways to deliver the messages depending on the application demands of the reliability, QoS, and semantic format.
- **Middleware layer:** This layer is responsible for performing the tasks of deployment, management, control and discovery of global computing environment resources. Moreover, it is in charge of arranging the process of requesting and responding services by and to the consumer.
- **Network Layer**: This Layer is responsible for routing and addressing of the message from one host to the other host.
- **Data-link layer:** This layer is responsible of transferring the message from one node to adjacent node over a link.

- **Physical layer:** This layer provides a communication system through guided and unguided media with encoding system to transfer messages between the nodes of the environment.

6.2.1. Middleware Layer

As indicated above, middleware carries out amongst other traditional middleware core services (Appendix A) resources management to provide high-availability, reliability and QoS management. As illustrated in Figure 6.2, the middleware layer is divided into three sub layers namely; *Serviceware layer*, *core-functions layer* and *resources overlay layer*.

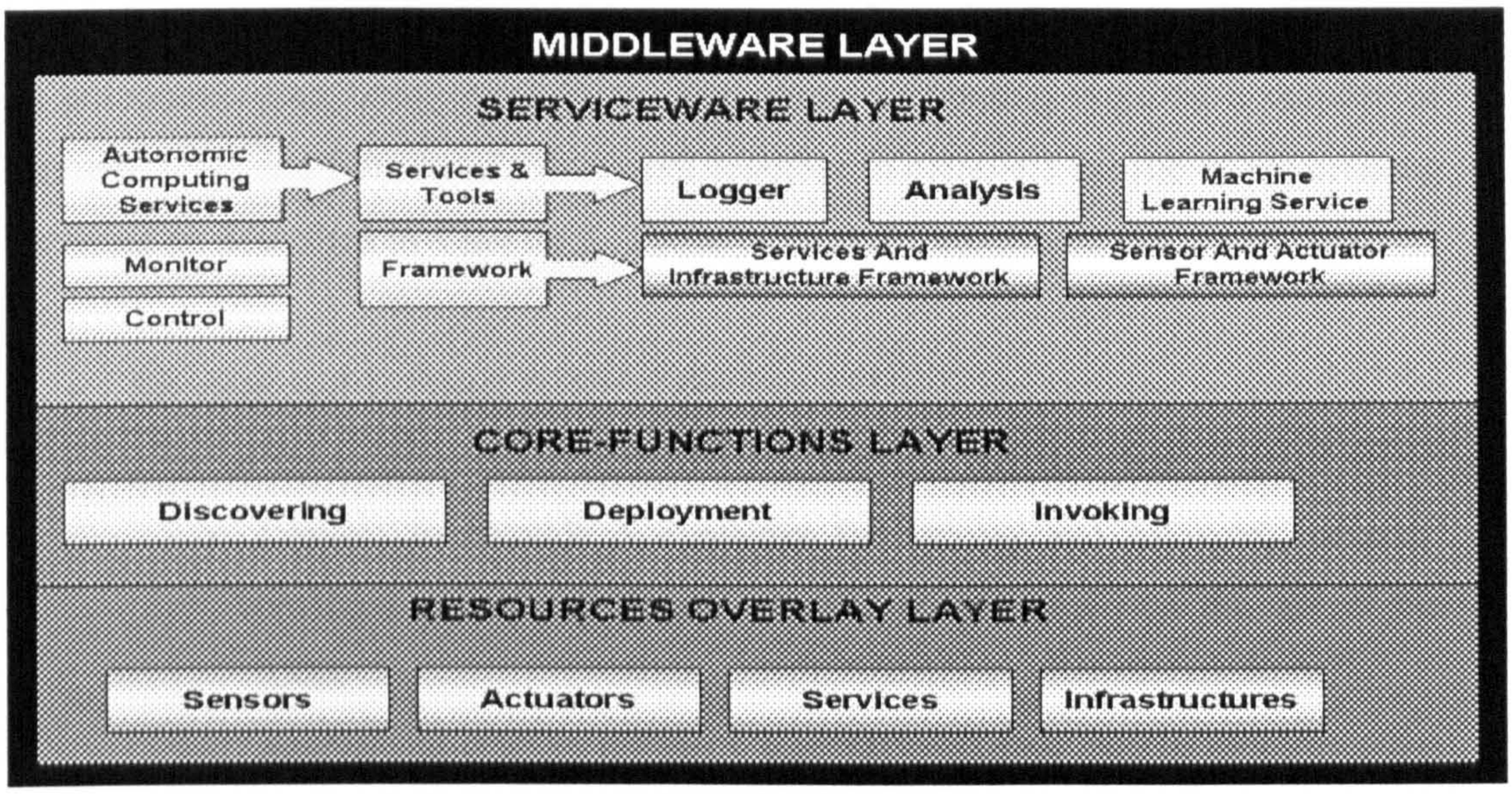


Figure 6.2: Middleware Layer

6.2.1.1. Serviceware Layer

Serviceware layer is responsible for a variety of decision tasks and jobs, such as:

- Managing, monitoring, controlling and adjusting or tuning the resources in order to provide highest availability, reliability, and QoS.
- Managing, monitoring, and controlling consumers' requests and system's responses to provide higher security system.

- Translating request and response message into semantic format to be understood by all actors of the system and to achieve the open standard concept.
- Taking an action in case of resources failure.
- Controlling the exchange of information with the other middleware systems in different clouds.

In addition to the above-mentioned tasks, Serviceware should support service-oriented programming by offering services such as monitoring, controlling and managing applications. Serviceware layer depends on variety of services, tools and frameworks to carry out the aimed tasks. Autonomic computing services are adopted in this layer as one of the core components to perform the required tasks depending on autonomic computing capabilities. Autonomic computing required a number of services, tools and frameworks that work together to complete the job of the creating decision, as described in Chapter 5. These tools and frameworks can be summarised as:

- **Services and Utilities:** different services and tools are required to complete the cycle of the autonomic services. The core services and tools are:
  - *Logger:* to provide a container for storing and retrieving sensors' readings in a standard and semantic format based on the use of the XML.
  - *Analysis service:* to present a way for analysing data in order to assets the intelligence services.
  - *Intelligence services:* to act as a brain for other tools, services and frameworks. Machine learning, data mining and statistics analysis are used to attach intelligent ability to these services. Other middleware functions, tools and frameworks offer a precious data and adjustment tools to the intelligent service. Intelligence services are responsible for doing intelligent classification, regression, analysis and prediction processes which help in making decisions.
- **Frameworks:** A variety of frameworks are required to provide a base for offering containers, information and recourses to the intelligence services. On

the other hand, the benefit to the frameworks from the functions, tools and services provided by the middleware functions and autonomic computing is expressed through managing their resources. Such frameworks are:

- *Assembly Services and Infrastructure Framework (ASIF):* to provide an overlay for intelligence services to do their tasks. It includes different sections namely; assembly services layer, services layer, and infrastructure layer.
- *Sensor and Actuator Framework (SAF):* to provide a foundation for deploying, discovering, and invoking of variety of software instrumentations (sensors), actuators and other monitoring resources.

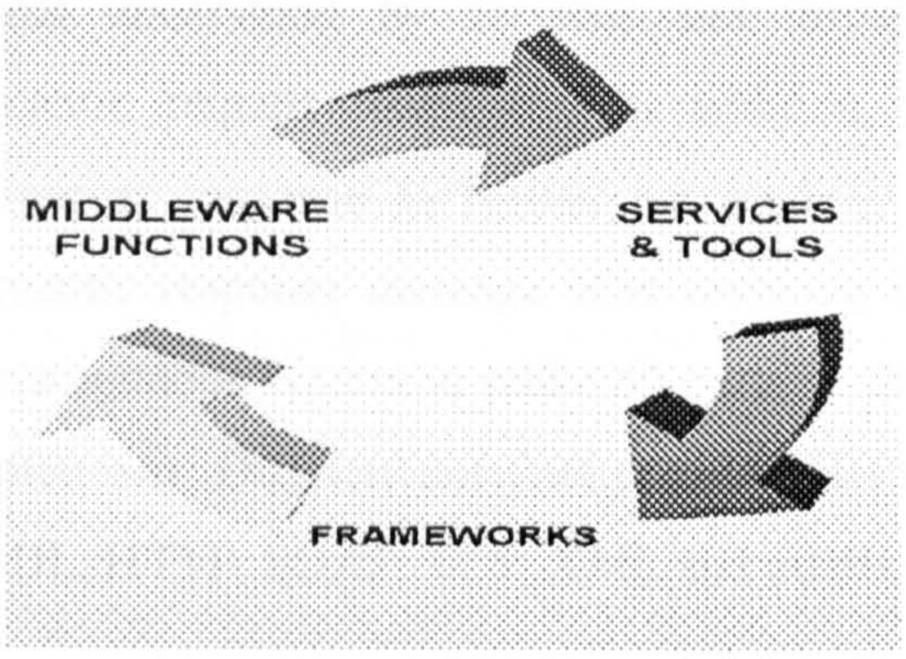


Figure 6.3: Framework Life Cycle

6.2.1.2. Core-Functions Layer

Core-functions layer consists of three main core functions responsible for joining the Serviceware and resources overlay layers. This layer is composed of three main core functions:

- **Discovery function:** works as a broker between consumers and middleware resources container. It is responsible for finding the most suitable services and/or infrastructures taking in consideration consumer's requests and SLA. The response of this function to the consumers' demands can be enhanced by relying on the collaboration between the autonomic services and discovering services. The discovering service provides the fabric of the semantic information, while autonomic service acts as a brain for doing intelligent staff

of analysing, deciding and selecting the most appropriate services which match consumers' requests.

- **Deployment function:** works as a broker between resources providers and resources container. The providers can deploy variety types of services like business, research, commercial, government, education and even autonomic services with resources container. Managing resources inside the resources container by the middleware is a necessity to reduce the required response time to accomplish to the demanded resources.

- **Invocation function:** works as a broker for translating consumers' requests to semantic format understood by services and infrastructures and hence generated semantic request message. In addition, it is responsible of translating resources response to format accepted by the consumers' and generating semantic response message after invoking the resources. This is valuable in heterogonous system to make all parts of the system talk to each others in common way. Different standard protocols are used to carry out such tasks, like WSDL, HTTP, SOAP, TCP, XML and others.

6.2.1.3. Resources Overlay

Resources overlay layer consists of all services and infrastructures required to serve the consumers in addition to the resources required to perform the basic tasks of the system such as networking and communication, monitoring and controlling. The resources container inside the resources overlay layer is proposed to be distributed over the system. Serviceware layer is responsible for providing control and monitor services to these distributed services. ASIF are proposed t represent the resources overlay.

6.3. Self-Management Software Design Pattern

The proposed self-management software design pattern language including the design of the Impromptu Framework [93] have been influenced by the Viable System Model [95, 96] – a general cybernetic management model. Thus it is important to give a

brief overview of the model before describing our design pattern language (Sec. 6.3.2).

6.3.1. VSM Model: a Brief Overview

Beer's Viable System Model defines the five systems that must exist for any entity to survive in a changing environment. The model explicitly incorporates dynamic planning and self-awareness systems, while the classical cybernetics that underpins the model is closely related to control systems theory. Beer describes the management of knowledge as follow [135]:

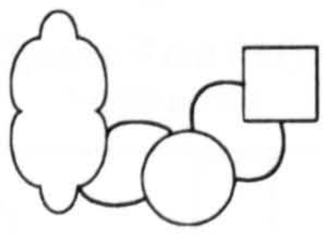
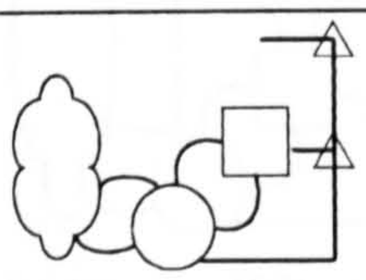
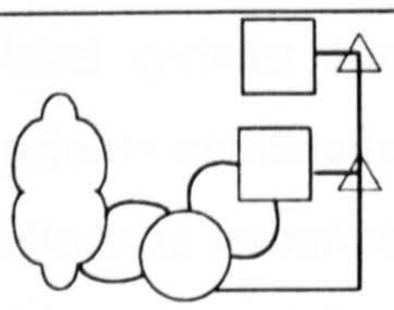
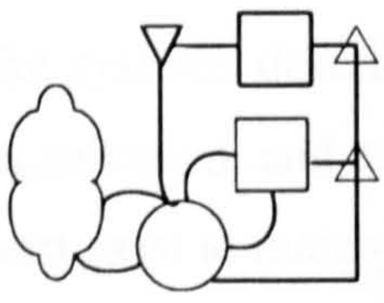
> *"...You must learn to manage yourself and your formal and informal exchanges and interactions with others. This must be done in the context of your understanding of who you are: your goals, your capabilities, your knowledge of your own strengths and weaknesses; and your appreciation of your social, technical and business environments. Individuals must be able to engage in activities in different 'markets', keep them from interfering with each other, manage them together, focus an eye on the future, and assess their different aspects from the perspective of the 'big picture' of their whole life's narrative..."*

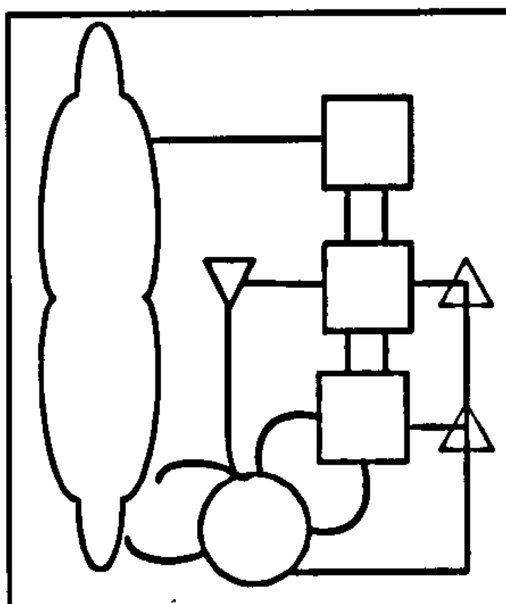
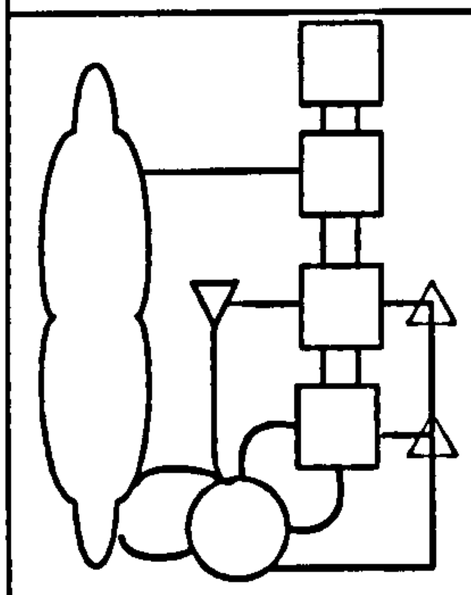
Although, Beer was talking about individuals, computing systems, which have the challenge of maintaining continuity and identity over time – sometimes with minimal infrastructure. They too must integrate and manage their knowledge and information and their exchanges with their environments to perform effectively. The VSM has been shown by Laws *et al.* [88, 90] to provide a powerful descriptive and diagnostic tool to map management capacities to promote viability.

The model identifies the necessary and sufficient communication and control systems that must exist for any organization to remain viable in a changing environment. In doing so, the model does not attempt to specify the activities that must occur in each system, instead activities are typified by cybernetic rational [88, 136] to allow either the designer of activities to match the cybernetic criteria or for actual activities to be identified by their system type and hence assigned to the appropriate element of the

model. Such a generalized approach allows the model to be applied to any organization regardless of size. The six major systems advocated by the model are detailed in Table 6.1 below:

Table 6.1: The Major Systems of Viable Systems Model [88].

System Identifier	System Type
 System One (S1) – Operations	System One performs the productive operations of the organization. An organization may be composed of a number System Ones, each providing a distinct product or service. Each S1 consists of an operational element controlled by a management process and in contact with the operational environment and is in some respects is similar to the plant/management arrangement adopted by control system theory.
 System Two (S2) – Coordination	System Two is concerned with coordinating the activities of S1 units. It is essentially anti-oscillatory in that it attempts to contain or minimize inter-S1 fluctuations. This is achieved by the provision of stabilizing, coordinating facilities such as scheduling and standardization information that is disseminated over all System Ones, but tailored locally to suit individual S1 needs.
 System Three (3) – Control	System Three is concerned with the provision of cohesion and synergy to a set of System One units. The management processes contained within this system will be concerned with short term, immediate management issues, such as resource provision and strategic plan production, although strategic in this context refers to planning with existing resources rather than the normally accepted sense.
 System Three* (S3*) – Audit	System Three * provides facilities for the intermittent audit of System One progress and provides direct access to the physical operations of the particular S1 allowing immediate corroboration of that progress. This essentially provides additional data over and above that provided by normal reporting procedures.

 System Four (S4) – Intelligence	System Four is concerned with planning the way ahead in the light of external environmental changes and internal organizational capabilities. S4 'scans' the environment for trends that may be either beneficial or detrimental to the organization and constructs developmental organizational plans accordingly. To ensure that such plans are grounded in an accurate appreciation of the current organization, the intelligence function contains an up-to-date model of organizational capability.
 System Five (S5) – Policy	System Five determines the overall purpose of the organization i.e. defines the activities that are performed by S1 as such S5 represents the policy-formulation or normative planning function. Policy formulation is informed by a "world-view" provided by S4 and models of current organizational capability populated by data flowing from the lower level systems in the organization.

The major systems (S1, S2, S3, S4 and S5) are structured hierarchically and connected by a central 'spine' of communication channels passing from the higher-level systems through each of the S1 management elements. These provide high priority communication facilities to determine resource requirements, accounting for allocated resources, alerts indicating that a particular plan is failing and re-planning is necessary and the provision of the "legal and corporate requirements" or the policies of the organization.

The systems described above concern the management structure a one level of the organization, and consequently specify the communication and control structures that must exist to manage a set of S1 units. However the power of the model derives from its recursive nature. Each S1 consisting of an operational element and it's management unit is expected to develop a similar SM-VSM structure, consequently, the structure of the systems is open ended in both directions and may be pursued either upwards to cover wider encompassing systems or downwards to cover smaller units. However, at each level the same structure of systems would occur although their detail would necessarily differ depending on context. This recursively allows

each level in the organization relative autonomy bounded by the overall purpose of the system as a whole.

To make the VSM model appropriate for planetary-scale software modelling and design VSM is here integrated with the Gang of Four (GoF) software design pattern [132] to design *Self-Management VSM* (SM-VSM).

6.3.2. SM-VSM Pattern for Self-Management System

Figure 6.4 illustrates the model for self-management viable system. The system is structured into five main layers as describes in the following paragraphs.

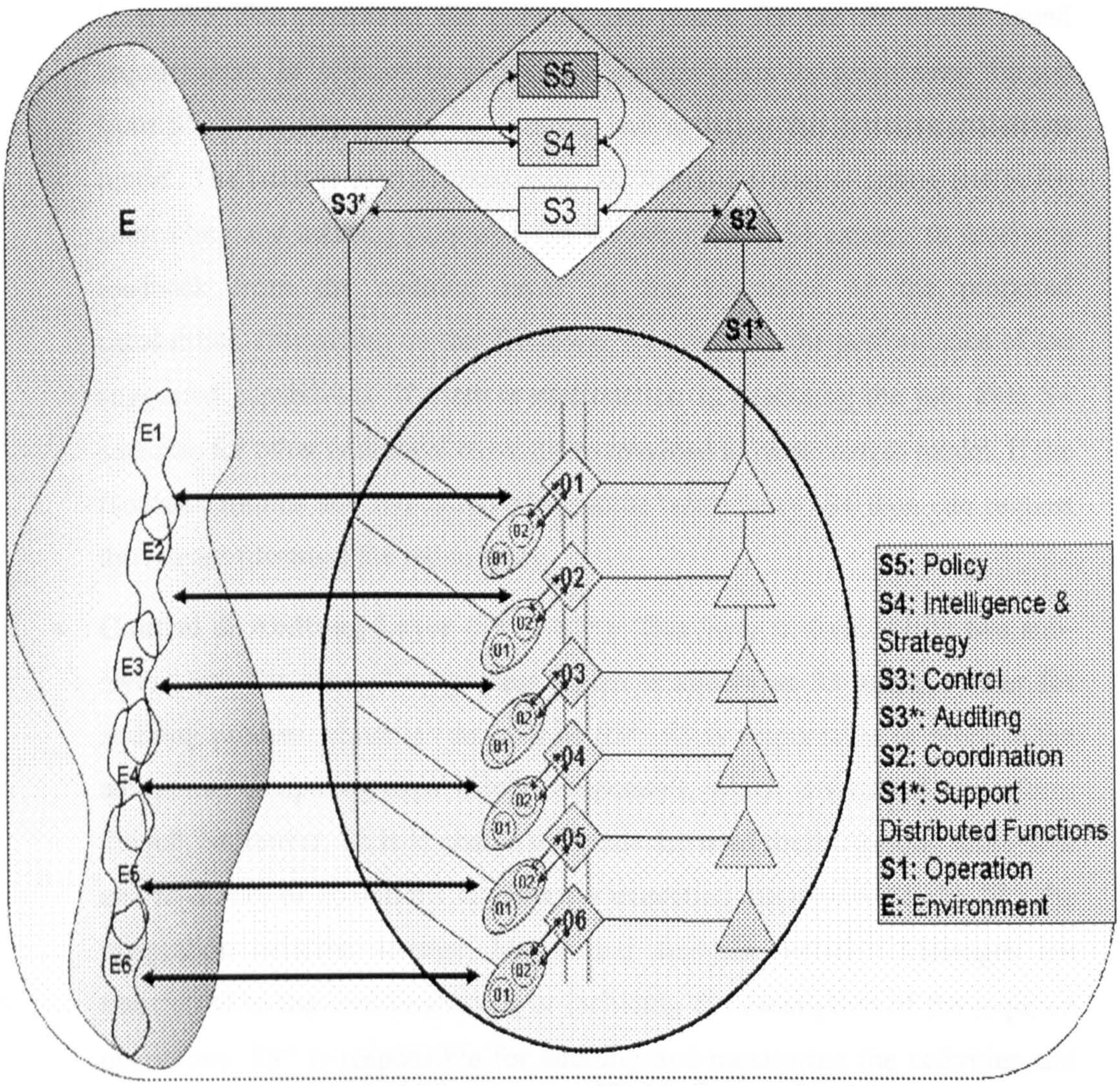


Figure 6.4: Self-Management Viable System Model

- **Policy Layer (S5):** This is the top layer of the SM-VSM model. It interacts with other S5 and generates management policies from Layer 4. It is responsible for analysing the demands from the lower layer (S4), and defining system policies and plans of action. Hence, it performs policy setting and action plan generation related to self-management tasks.

- **Intelligence Layer (S4):** This layer interacts with the environment in order to address the self-management requirements, which can be detected from scanning the environment such changing behaviour or anticipated/predicted need for self-management. These requirements are forwarded to the upper layer (S5) for generating the policies and planes for the above-mentioned requirements. In addition to receive the outline plans for performing the job from S5, S4 is responsible for forming and updating the system capabilities model. The latter is used for reasoning and forming a plan of action and/or for controlling and auditing purposes. On the other hand, S4 expects to receive a feedback from the auditing layer on the behaviour of the proposed capabilities. Depending on this feedback, S4 evaluates the performance of the suggested capabilities. If there is malfunction in executing the job, then S4 searches for other additional capabilities to added to its resources model. If the feedback returns and indication of resource redundancy, S4 then can request the release (destroy) the redundancy.

- **Control & Auditing Layer (S3 & S3*):** This layer is divided into two sub-layers, one for controlling process which is known as S3 and the other for auditing process which is denoted as S3*. S3 is responsible to propose the autonomic computing capability's components, which is required to perform the job. Moreover, S3 is in charge to decide the need for integrating more than one resources or operations together for executing the required tasks, like the integration between sensors, logger and analysis services. Messages are transferred to the coordination layer including the description of the required operations. S3* is responsible for auditing and monitoring the activities and behaviour of the running operations. S3* should be intelligent enough to

decide the action that is required according to the auditing process. The action can be represented as feedbacks to S4 or as notify messages to S3 for adding or deleting operations. In addition, S3* is responsible for monitoring and controlling the corporation and exchange of information between the autonomic capabilities services and their components at the operation level.

- **Coordination Layer (S2):** This is responsible for selecting the phase of operation (deploy, discover, and/or invoke). This can be achieved by describing the status of each operation as *Start, Stop* and *Wait*. "Start" status indicates to the initiating of the one of the above-mentioned phases for specific operation. "Stop" status refers to the end of the phase. "Wait" status refers to the need for waiting another action to be taken before requesting phase for the specific operation. The requested phase is sent to the lower layer.

- **Operation and Support Distributed Function Layer (S1 & S1*):** This is the lower layer of the SM-VSM. It is divided into two sub-layers, namely; *Operation* (S1) and *Support Distributed Function Layer* (S1*). This layer is responsible for performing a given operation, and for middleware/distribution basic functions respectively. For instance, S1 can represent operational unit to undertake both functional and non-functional systems requirements including autonomic computing capabilities. In line with programming constructs each S1 sub layer has its own private and public operations container, which we refer to as *private and public operation containers* respectively. The public operation container includes all the operations deployed by different providers, while the operations in the private resources container is belong to one system and can not be employed by other system. S1* is responsible for implementing the core functions of calling and utilising operations in large-scale system. This consists of discovering, deploying and invoking operations as have been described in next sections. Moreover, S1* includes other support services like sensors, actuator and logger for the auditing process. S1, on the other hand, executes the operations.

- Environment (E): This represents the consumer who asks for the self-management service. The Environment judges the quality of the tasks offered by SM-VSM. If the environment discovers the QoS is not satisfactory, then it has the right to ask for another self-management system for executing the job.

6.3.3. Self-Management Viable System Scenario

The system model for self-management pattern consists of three actors, as shown in Figure 6.5, namely; *environment, public operations container*, and *private operation container*. Moreover, the model consists of five use cases, which are: *policies, intelligent and strategy, control, coordination* and *support distributed functions*. These use cases represent the layer of the SM-VSM model.

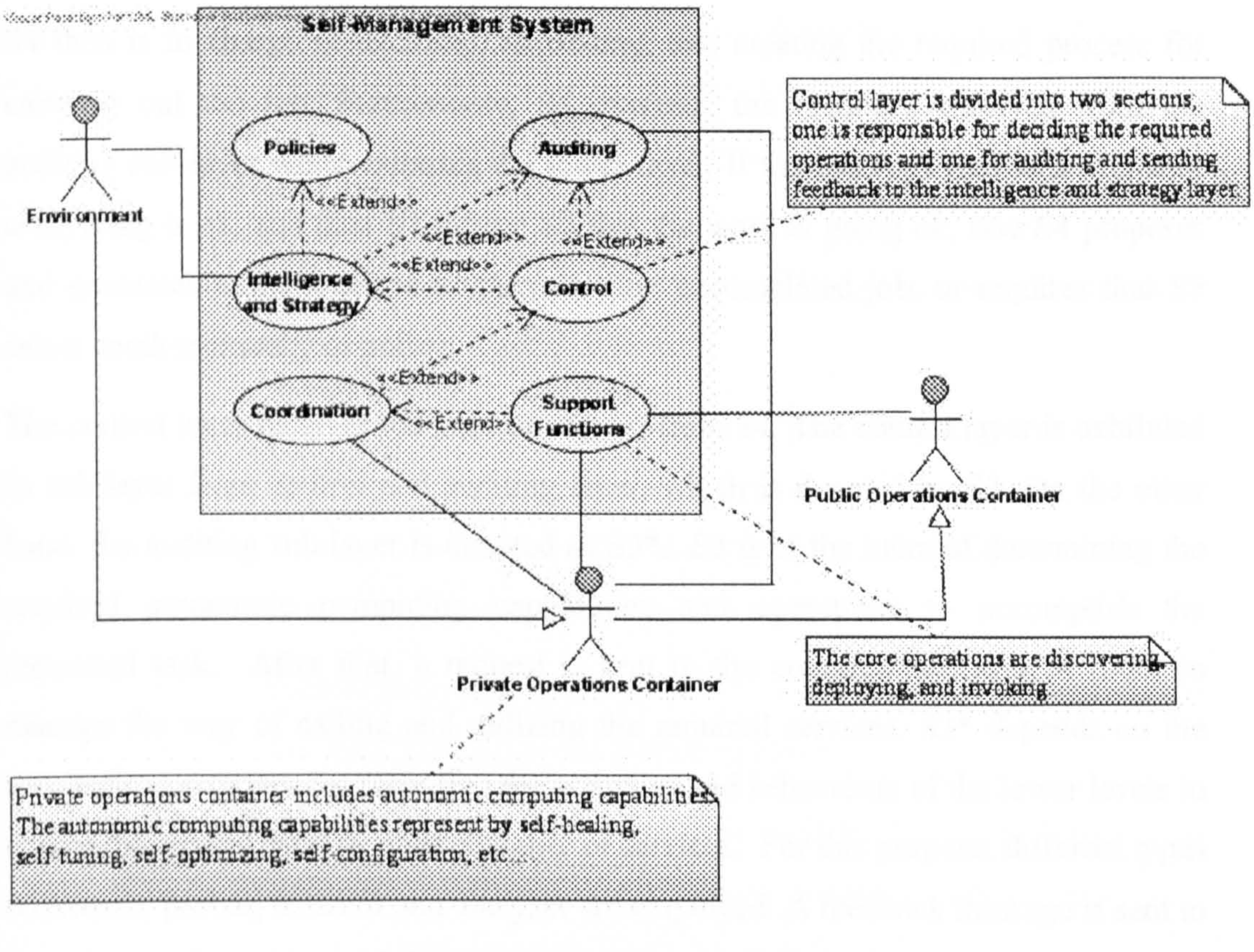


Figure 6.5: SM-VSM-UML Use Case Diagram

The process initiates from the environment by requesting a self-management service, as shown in the sequence and activity diagrams in Figures 6.6 and 6.7 respectively. Moreover, the self-management service can be triggered according to the changes in

the environment nature or behaviour. Therefore, the system tries to generate a pattern that is responsible for recovering the unexpected behaviour. Intelligence layer (S4) is responsible for receiving the request from the environment or scanning the environment for detecting any unexpected activities in its behaviours. Then, S4 passes a message with the required requests to S5 in order to find out the required policies.

The policy layer analyses these parameters (requests or unexpected behaviour) in order to sort out the way of generating the required patterns and operations. Policy in this case represents S5 of the SM-VSM model. S5 commences planning the outline strategies for performing the requested tasks. These blueprints are forwarded to the intelligence and strategy layer, (which is S4 in the SM-VSM).

S4 then is in charge of deciding, requesting, and creating the required process for carrying out the job. Furthermore, S4 oversees the feedback received from the auditing sub-layer which controls the lower level. If the received feedback indicates everything is ok, and then S4 inform S5 that the work is going ok, else S4 proposes and generates new processes to carry out the uncompleted job, or requires that S5 select another strategy or policy.

The control layer receives the process request from S4. The control layer is exhibited as sub-layer from control and auditing layer, which is denoted by S3. On the other hand, the auditing sub-layer is denoted as S3*. S3 is at the helm of determining the required autonomic computing capabilities and operations to accomplish the requested task. After that, a request is sent to the coordination layer in order to manage the way of calling and utilizing the required services. S3* depends on the monitoring resources for recording the activities and behaviours of the lower levels to ensure the quality, fidelity and reliability of services. For this purpose, different types of sensors, probes, actuator and analyser are employed. A feedback message is sent to S4 in case of unexpected behaviour of the lower layers, or failure of the demanded jobs.

At this point, a request of required operations is despatched from S3 to the coordination layer (represented as S2 in the SM-VSM model). S2 supervises the

process of discovering operations in the public operation container, deploying the discovered operations with the private operation container, and tracking the status of operations. The statuses of operations are indicated by:

- **Start:** to start the process by the selected operation.
- **Stop:** to end the process in case of the end session or unexpected behaviour.
- **Wait:** to wait an action from other operations.

For the above objectives of utilising operations, three support distributed functions and operations should be accessible for any self-management model. These core functions or operations are: discovery, deployment and invocation. Therefore, we presume these operations are at an intermediate level between the coordination and operation layers. We call this intermediate level the *Support Distributed Functions Layer* which we denoted as S1*. For each requesting operation, three messages transfer from S2 to S1* and one message from S2 to S3. These messages are explained in the following points:

- The first transfer message from S2 to S1* is the request to locate and find operations in the public operation container.
- The second transfer message from S2 to S1* is the request to deploy operations from the public operation container to the private operation container. This message is sent after selecting the most appropriate operation from the discovered operations in the public container.
- The third transfer message from S2 to S1* is the request for invoking the operation after transferring it to the private container.
- There is another notify message transfer from S2 to S3. This "notify" message indicates the success or failure of generating the requested operations.

To this end, the private operations container is populated with the required operations to carry out the task. Moreover, full information describing the invocation of the operation and interaction between the operation and environment is available at S1*. The private operations container is located at the operation layer which is symbolised

as S1. ASIF and SAF are designed, developed and implemented to support the deploying of services and operations in public operation container.

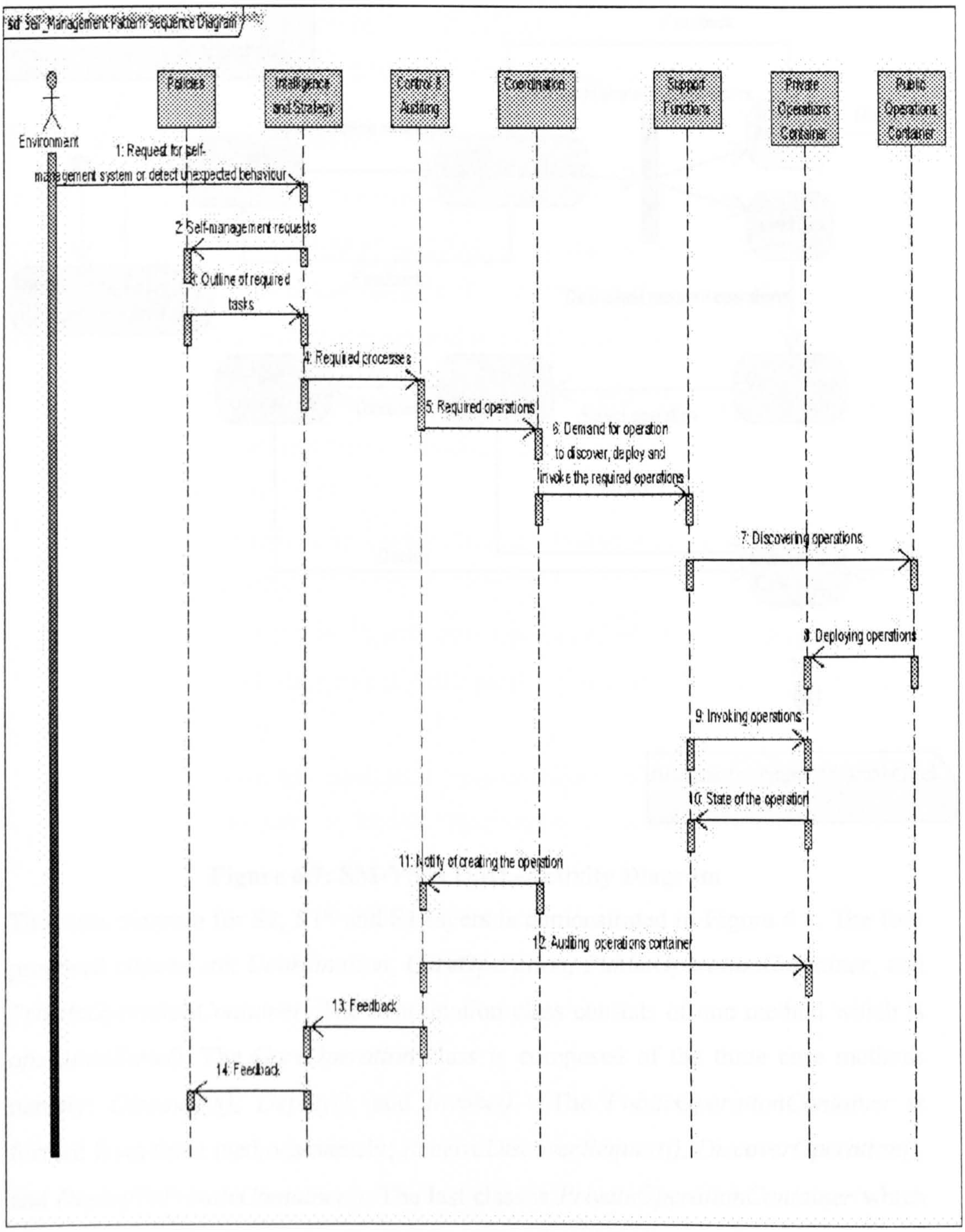


Figure 6.6: SM-VSM-UML Sequence Diagram

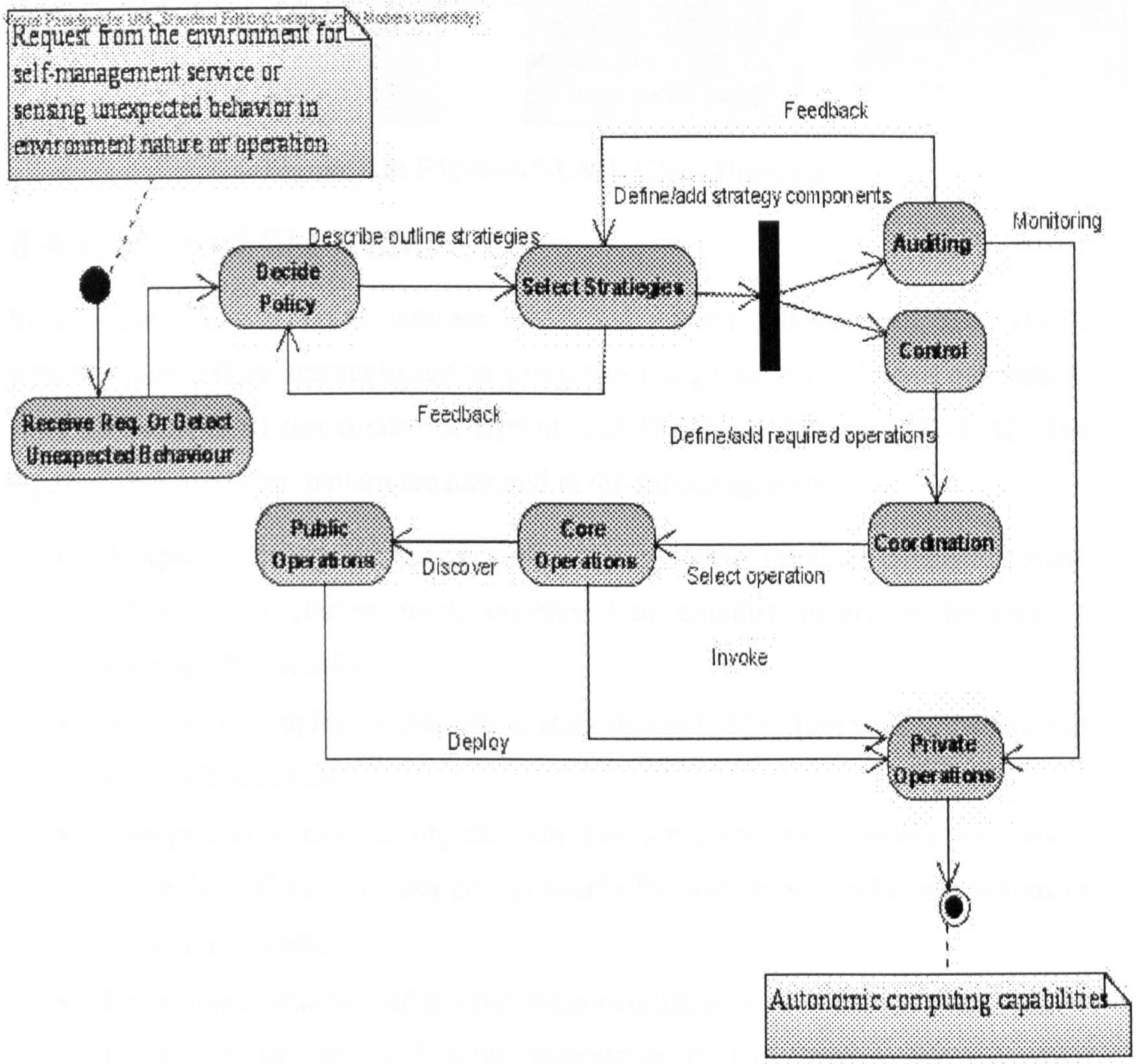


Figure 6.7: SM-VSM UML-Activity Diagram

The class diagram for S2, S1* and S1 layers is demonstrated in Figure 6.8. The four proposed classes are: *Coordination*, *CoreOperation*, *PublicOperationContainer*, and *PrivateOperationContainer*. The coordination class consists of one method which is *operationState()*. The *CoreOperation* class is composed of the three core methods namely; *Discovery()*, *Deploy()*, and *Invoke()*. The *PublicOperationContainer* is formed from three methods namely; *ReceiveDiscoverRequest()*, *DiscoverOperation()*, and *DeployToPrivateContainer()*. The last class is *PrivateOperationContainer* which includes two methods namely; *ReceiveOperation()*, and *DoProcess()*.

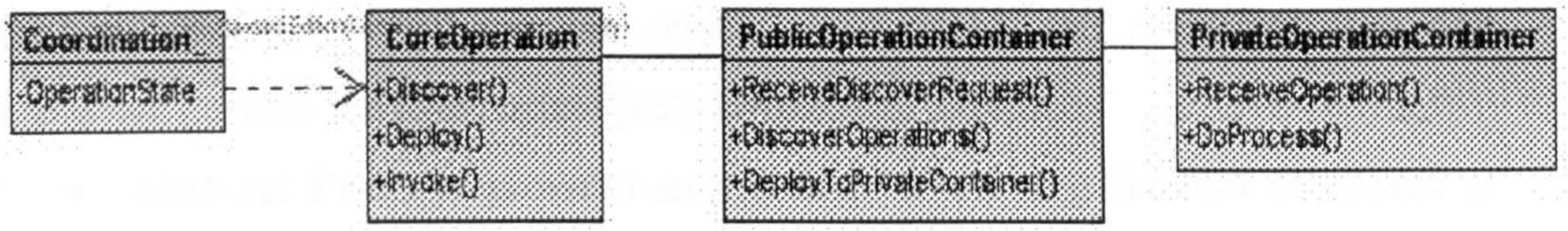


Figure 6.8: SM-VSM-UML Class Diagram

6.4.GoF and SM-VSM

In our research, a mapping between SM-VSM pattern and Gang of Four (GoF) pattern [132-134] is accomplished to bring the design to the software generation. Each operation in S1 represents one type of *GoF Structural Patterns* [132-134]. The types of GoF structural pattern are narrated in the following points:

- **Adapter:** converts the interface of a class into other interface clients expected. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.
- **Bridge:** decouples an abstraction from its implementation so that the two can vary independently.
- **Composite:** composes objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
- **Decorator:** attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub-classing for extending functionality.
- **Façade:** provides a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use.
- **Flyweight:** uses sharing to support large numbers of fine-grained objects efficiently.
- **Proxy:** provides a surrogate or placeholder for another object to control access to it.

The full details of the structural pattern are found in Appendix B. Structural patterns composing classes to form larger structures. Such classes are generated by the *Creational Patterns of GOF* [132-134]. Creational model makes systems independent

of how operations or objects are created, represented, and composed. The five creational patterns are listed below [132-134].

- **Abstract Factory:** provides an interface for creating families of related or dependent objects without specifying their concrete classes. The objects in this case can be represented as service, infrastructure, control unit, monitor unit (sensor or actuator), and intelligent service.
- **Factory Method:** defines an interface for creating an object according to subclasses demands in which class to instantiate. For example, in sensor case, the factory method is responsible for defining the required methods for each type of sensors.
- **Prototype**: specifies the kind of objects that required to be created using a prototypical instance, and create new objects by copying this prototype. In our example, the system specifies the new objects as virtual memory sensor and creates it by copying depending on prototypical instance.
- **Singleton:** ensures a class has only one instance and provides a global point of access to it.

The following section introduces examples of using GoF creational patterns with SM-VSM model for self-management system.

6.4.1. Illustrative Examples

In this section, varieties of examples are introduced to illustrate the usage of the GoF creational patterns with SM-VSM model. Self-Tuning (self-optimising) operation as S1 with its required components is selected as examples to show the idea.

6.4.1.1. Abstract Factory

Abstract factory from GoF creational patterns is used to design a pattern for requesting self-tuning capability as an example of demanding autonomic computing capabilities. The abstract factory consists of five main classes as shown in the UML class diagram in Figure 6.9. Figure 6.10 demonstrates the code in C# for implementing this operation. The classes are:

- **AbstractFactory** (Self-Tuning)
    - declares an interface for operations that create abstract products
- **ConcreteFactory** (High QoS, Load Balance)
    - implements the operations to create concrete product objects
- **AbstractProduct** (Replication, mirror, load distribution)
    - declares an interface for a type of product object
- **Product** (Sensor, Actuator, ML, Discovery, Deploy, Invoke, Logger)
    - defines a product object to be created by the corresponding concrete factory
    - implements the *AbstractProduct* interface
- **Client** (S1 Operation) Coming from structural behaviour
    - uses interfaces declared by AbstractFactory and AbstractProduct classes

The complete code for creating the self-tuning capability is shown in Figure 6.10.

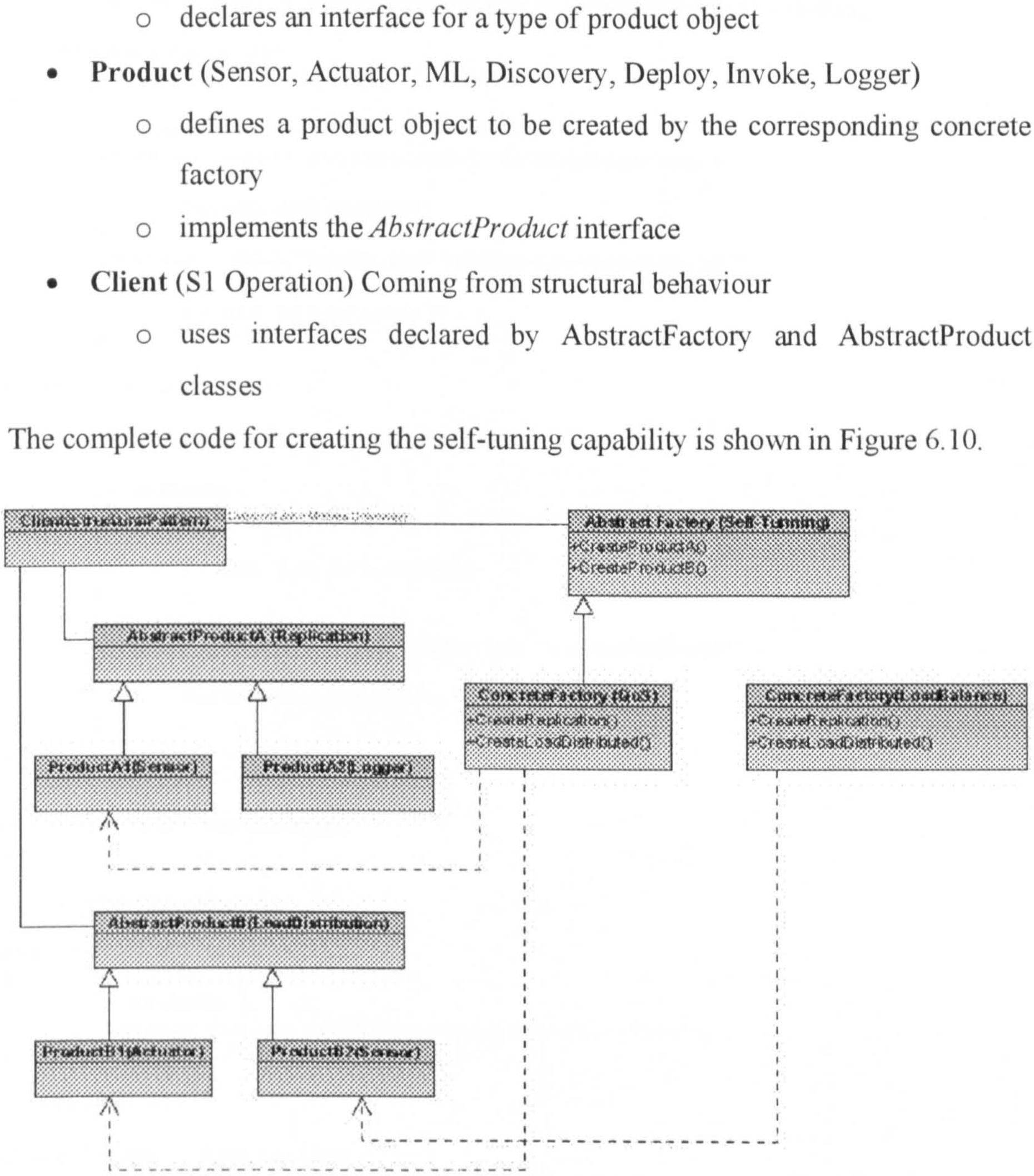


Figure 6.9: GoF Abstract Factory Pattern

```
// Abstract Factory pattern -- Structural example

using System;

// "AbstractFactory"

abstract class SelfTuning
{
        // Methods
        abstract public Reiplication CreateReplication();
        abstract public LoadDistribution CreateLoadDistribution();
}
// "ConcreteFactory1"
class QoSFactory : SelfTuning
{
        // Methods
        override public Replication CreateReplication()
        {
                return new Sensor();
        }
        override public LoadDistribution CreateProductB()
        {
                return new Logger();
        }
}
// "ConcreteFactory2"

class LoadBalanceFactory : SelfTuning
{
        // Methods
        override public Replication CreateProductA()
        {
                return new Actuator();
        }

        override public LoadDistribution CreateProductB()
        {
                return new Sensor();
        }
}
// "AbstractProductA"

abstract class Replication
{
}
// "AbstractProductB"

abstract class LoadDistribution
{
        // Methods
        abstract public void CreateCopy( Replication a );
}

// "ProductA1"

class Sensor : Replication
{
}
// "ProductB1"
class Actuator : LoadDistribution
{
        // Methods
        override public void CreateCopy( Replication a )
```

```
        {
                Console.WriteLine( this + " Create Copy " + a );
        }
}

// "ProductA2"

class Logger : Replication
{
}

// "ProductB2"

class Sensor : LoadDistribution
{
}

// "Client" - the interaction environment of the products

class SelfManagement
{
        // Fields
        private Replication Replication;
        private LoadDistribution LoadDistribution;

        // Constructors
        public SelfManagement( SelfTuning factory )
        {
                Replication = factory.CreateReplication();
                LoadDistribution = factory.LoadDistribution();

        }

        // Methods
        public void Run()
        {
                LoadDistribution.Interact( Replication );
        }
}

/// <summary>
/// ClientApp test environment - Autonomic Computing Capability
/// </summary>
class ACCapability
{
        public static void Main(string[] args)
        {
                AbstractFactory QoS = new QoSFactory();
                SelfManagement SM1 = new SelfManagement( QoS );
                SM1.Run();

                AbstractFactory LoadBalance = new LoadBalanceFactory();
                SelfManagement SM2 = new SelfManagement( LoadBalance );
                SM2.Run();
        }
}
```

Figure 6.10: C# Code for Generating an Abstract Factory for Self-Tuning Capability

6.4.1.2. Builder

Builder pattern from GoF creational patterns is used to design a pattern for demanding sensors from monitor class. The builder pattern is demonstrated in UML class diagram in Figure 6.11. The components of the builder pattern are:

- **Builder** (Software Sensor)
    - specifies an abstract interface for creating parts of a Product object
- **ConcreteBuilder** (Load Balance Sensor, Bandwidth Sensor)
    - constructs and assembles parts of the product by implementing the Builder interface
    - defines and keeps track of the representation it creates
    - provides an interface for retrieving the product
- **Director** (Monitor)
    - constructs an object using the Builder interface
- **Product** (Memory Sensor, Transmit Rate Sensor)
    - represents the complex object under construction. ConcreteBuilder builds the product's internal representation and defines the process by which it's assembled
    - includes classes that define the constituent parts, including interfaces for assembling the parts into the final result

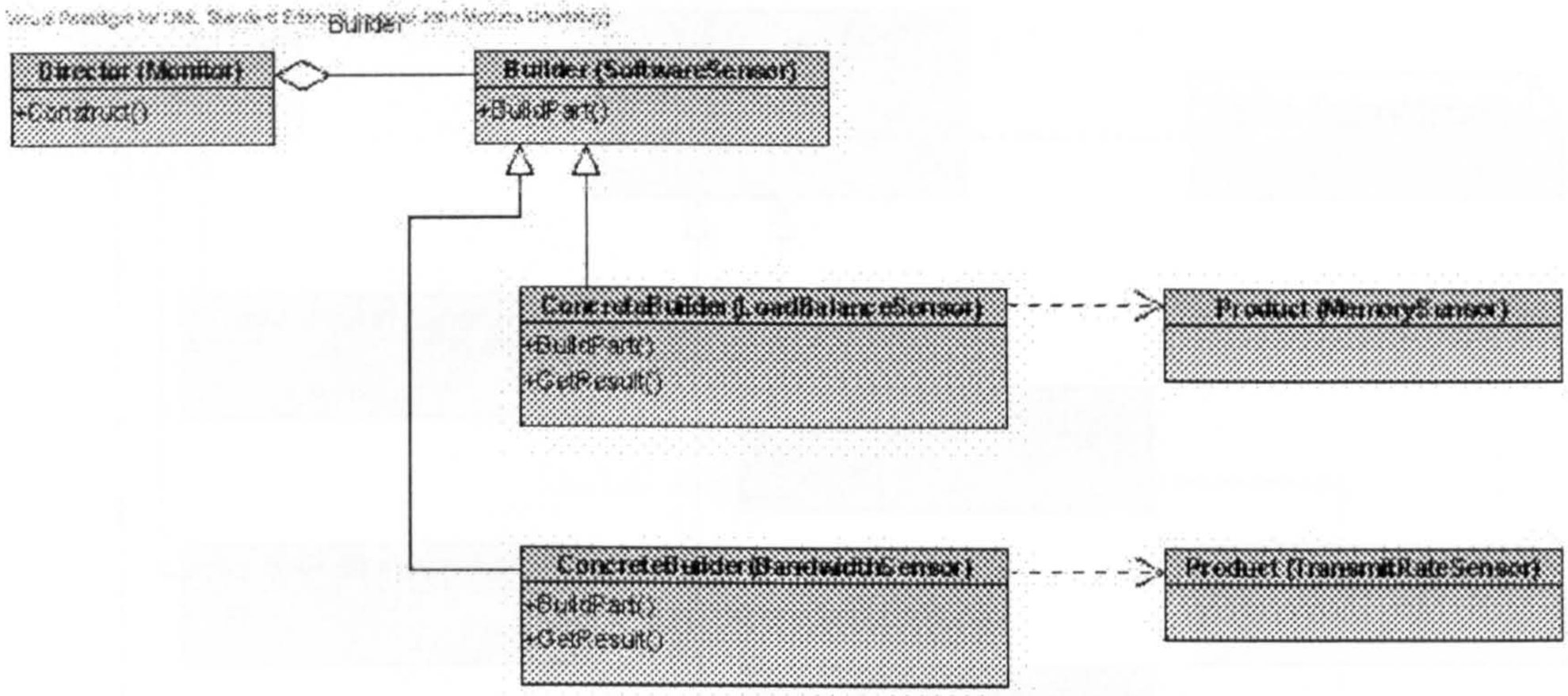


Figure 6.11: GoF Builder Pattern

6.4.1.3. Factory Method

Factory method from GoF creational patterns is used in this example to create different types of sensors. The factory method pattern is demonstrated in UML class diagram in Figure 6.12. The components of the factory method pattern are:

- **Creator** (Load Balance Sensor)
  - Declares the factory method, which returns an object of type Product. Creator may also define a default implementation of the factory method that returns a default ConcreteProduct object.
  - May call the factory method to create a Product object.
- **ConcreteCreator** (Memory Senor, Process Sensor,)
  - Overrides the factory method to return an instance of a ConcreteProduct.
- **ConcreteProduct** (Base memory, Virtual memory , Extended memory)
  - Implements the Product interface
- **Product** (Memory)
  - Defines the interface of objects the factory method creates

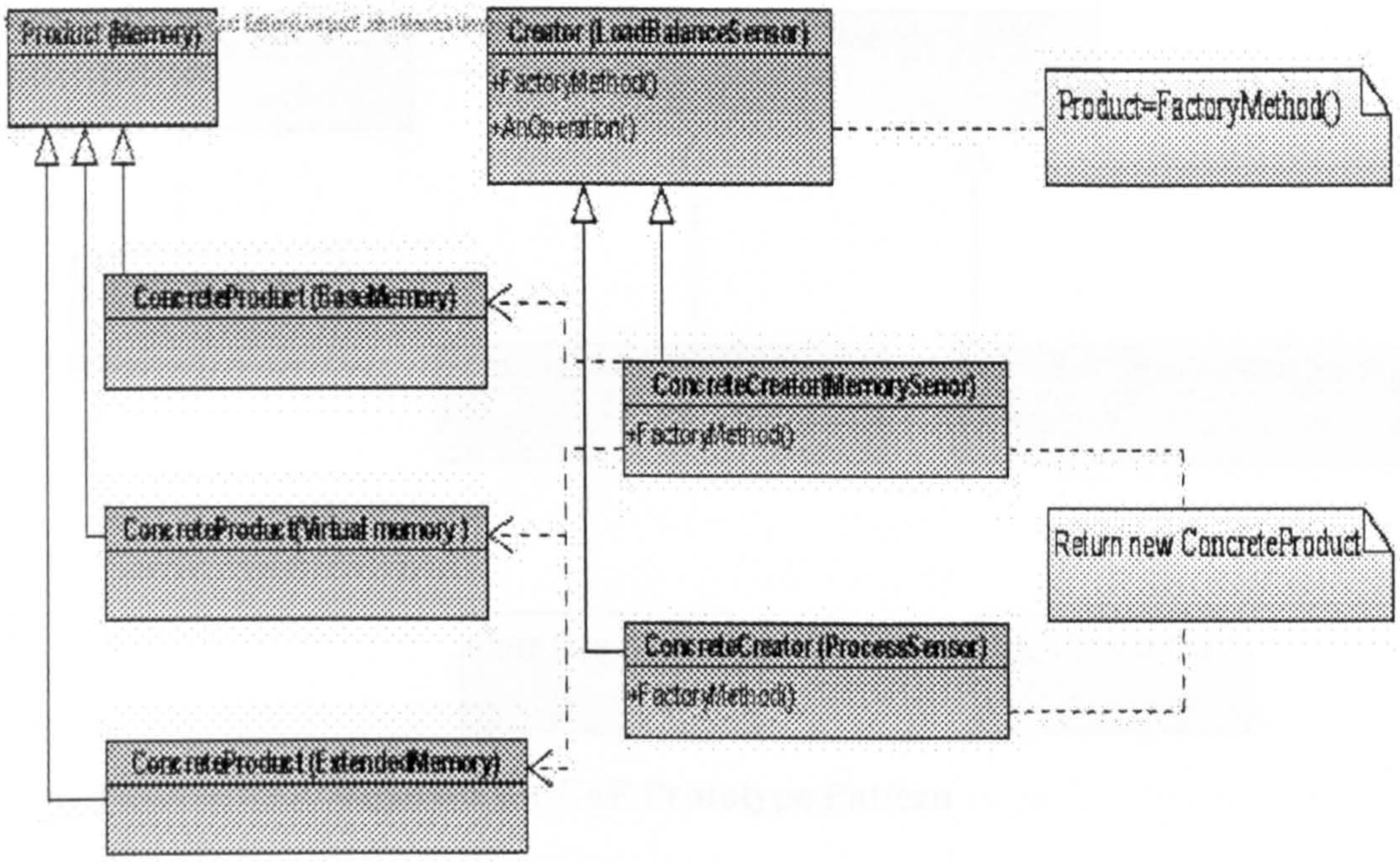


Figure 6.12: GoF Factory Method Pattern

6.4.1.4. Prototype

Prototype pattern from GoF creational patterns is used in this example to generate different types of memory sensors. The prototype pattern is illustrated in UML class diagram shwn in Figure 6-13. The components of the factory method pattern are:

- **Prototype** (MemoryPrototype)
  - declares an interface for cloning itself
- **ConcretePrototype** (Memory)
  - implements an operation for cloning itself
- **Client** (MemorySensorManager)
  - creates a new object by asking a prototype to clone itself

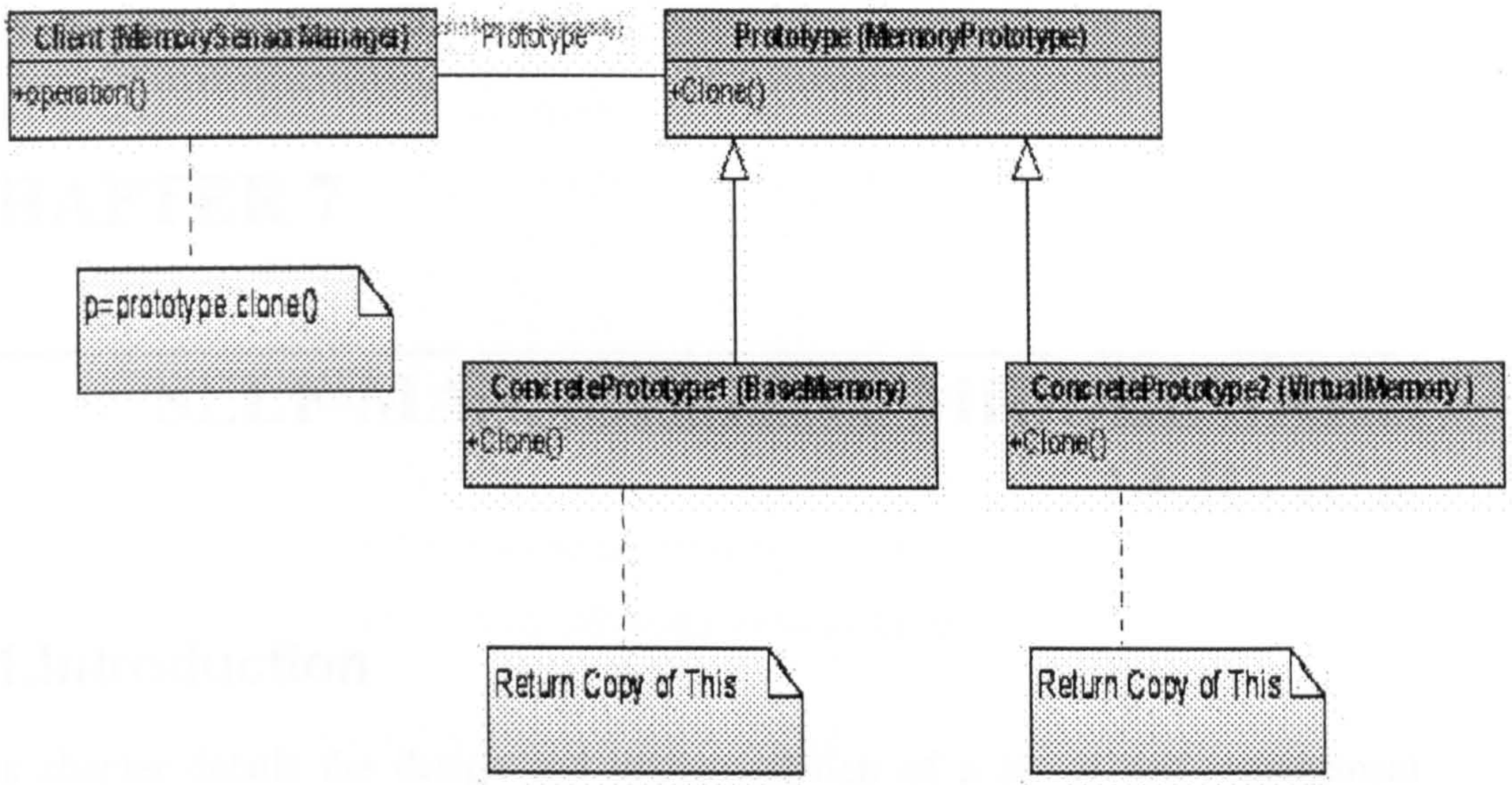


Figure 6.13: GoF Prototype Pattern

6.5.Summary

This chapter detailed a reference model including a design pattern for self-management software services for autonomic grid systems. The model is built by extending the Internet model with middleware layer between transport layer and network layer. The proposed middleware layer is in charge of controlling the access to the distributed resources by the consumers.

Based on the above-mentioned model, another model for viable self-management system is also presented through this chapter in order to generate a system that able to face and react to different unexpected changes and behaviour in the environment, take in the account the nature and boundaries of the environment applications. The proposed model is merged with a well known software pattern model in order to generate model accepted in the software community.

The proposed models require number of support tools to carry out the required management job from the distributed self-management system. Such support utilities are: intelligent web services, services and infrastructures framework, monitoring system and sensor and actuator framework. Next chapter discusses these support utilities.

CHAPTER 7

SELF-MANAGEMENT MIDDLEWARE SERVICE

7.1.Introduction

This chapter details the design and implementation of a set of self-management middleware services, utilities and frameworks (Sec. 5.2.3). In particular, the proposed self-management utilities are provided to support middleware tasks such as; intelligent prediction, classification, clustering and machine learning tasks. Such autonomic computing services are grouped in the Serviceware layer (Sec. 6.2.1), and are implemented as web services in compliance with the Open Grid Services Architecture (OGSA) specifications. In addition, two frameworks are proposed here to facilitate design-time and runtime assembly and deployment of application services and infrastructures (grids and/or overlays) together with sensor and actuator overlays for general monitoring and control services.

7.2.Intelligent Web Services Design

The following sections detail the development of two chosen examples of self-management middleware utilities namely; a *Self-Organising Map (SOM)* and *multiple regression analysis*.

7.2.1.Machine Learning Utility

As discussed in Section 4.2, in addressing the shortcomings of current policy-based autonomic systems design [15, 17, 99, 100], that is, the lack of support for policy (rules) evolution reflecting the most up to date state of a given system's management and operational model. An unsupervised machine learning algorithm is here used for

instance to systems usage prediction, usage classification and conditional triggers for self-management policies. Such models can then be accessible by the developed autonomic middleware control services for instance as shown by Badr [7] to control the self-healing processes.

7.2.1.1. SOM Implementation

SOM service is designed, developed and implemented as a web service to be used by the autonomic computing capabilities. Hence, it considered in this work as an autonomic middleware utility. The following sections describe the SOM algorithm.

7.2.1.2. Classification Method Using SOM

Self-Organising Maps (SOM) is one of unsupervised methods [137]. There are two basic groups of unsupervised learning algorithms related to self-organized neural networks, namely: (generalized) *Hebbian Learning* and *Competitive Learning* [138]. Figure 7.1, illustrates a Hebbian learning algorithm, where X, W, y and φ represent the input, weight, output and the activation function respectively. The SOM training and analysis can be summarized in the following steps [139]:

Step 0: Initialise weight $w_i(t)$.

Letting topological neighbourhood parameters $N_c(t)$.

Letting Learning rate parameters $\alpha(t)$ and $h_{ci}(t)$.

Step 1: For each input vector $x(t)$, do

a. Finding a BMU $\|x(t)-w_c(t)\| = \min_i \|x(t)-w_i(t)\|$

BMU is the best matching unit

b. Learning process

$$w_i(t+1) = \begin{cases} w_i(t) + h_{ci}(t)[x(t) - w_i(t)], i \in N_c(t) \\ w_i(t) \end{cases} \quad \text{.....(7.1)}$$

c. Going to the next unvisited input vector. If there is no visited input vector left, then going back to the very first one else going to step 2.

Step 2: Incrementally decreasing the learning rate and the neighbourhood size, and then repeating step 1.

Step 3: keeping doing step 1 and 2 for a sufficient number of iterations.

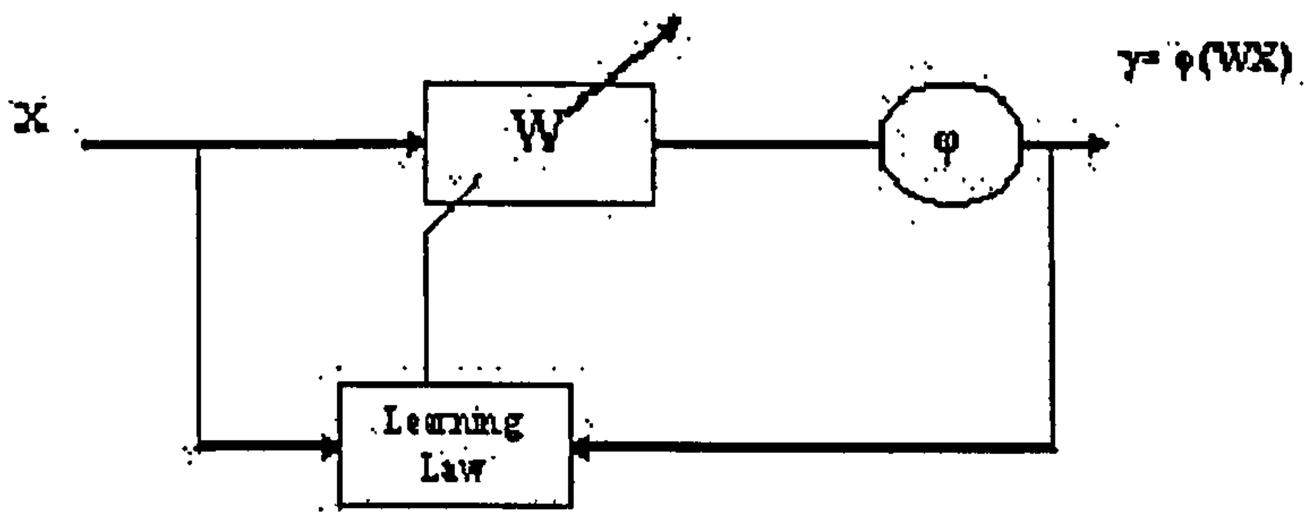


Figure 7.1: A Bock Diagram of a Basic Hebbian Learning Neural Network (SOM) [139]

As shown in Figure 7.2, the SOM is initialising using either random or linear initialisation. For train the map, SOM uses sequential or batch algorithms. The resulting visual map exhibits the neighbourhood between the neurons and the input training samples updating Best Matching Unit (BMU). The quantization error could be measured using *som_quality* [140] (from Matlab SOM toolbox) function which supplies two measures: average quantization error and topographic error. Figure 7.2 illustrates SOM lifecycle model, which delivers logic decisions from the visual maps taking benefit from labelling feature in *som_autolabel* and *som_addlabels* functions in Matlab library [140]. Hence we can build a programming model achieving SOM method and outputting decisions from calculating BMU for a given data vectors using *som_bmus* function and other related useful functions provided by the toolbox. The detail implementation of SOM using Matlab functions is presented in Sec 9.4.1.3.

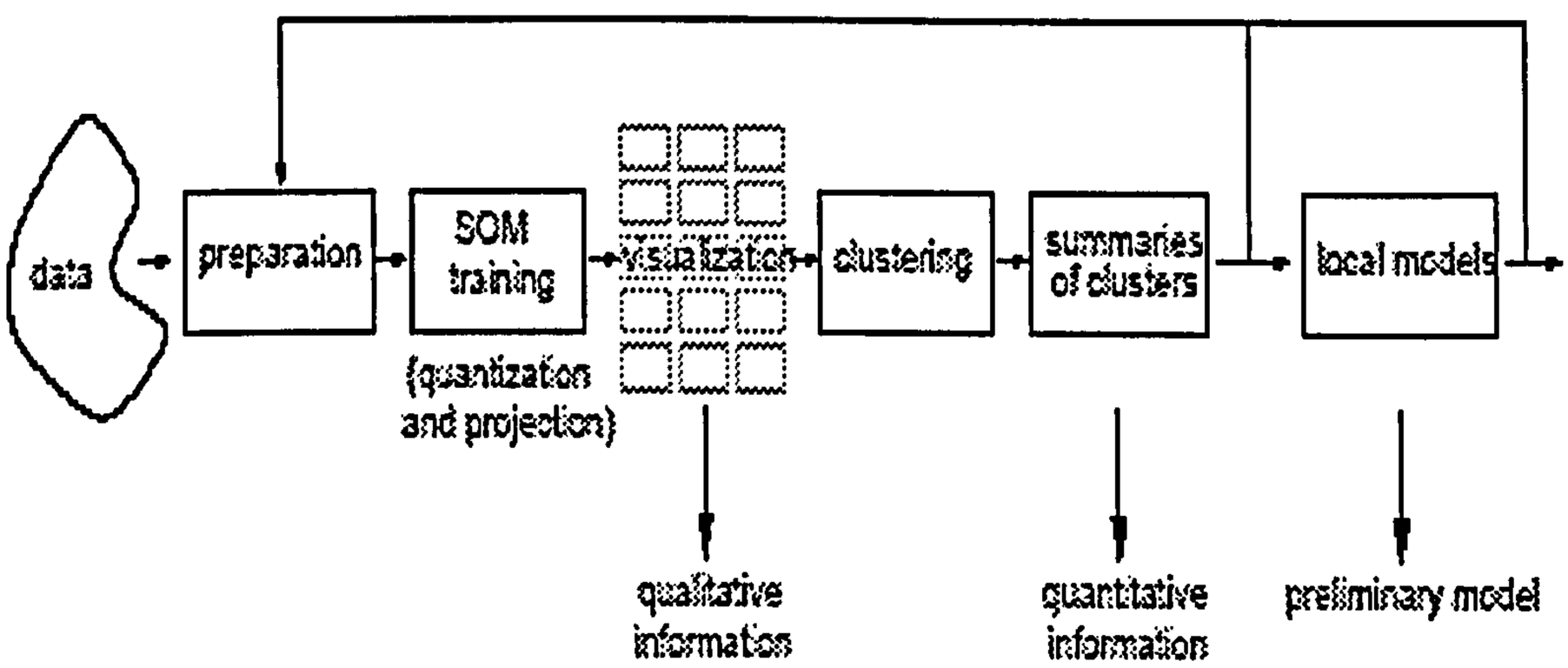


Figure 7.2: Using SOM in Preparation Survey Cycle [141]

7.2.2. Regression Analysis Utility

Multiple regression analysis [142, 143] is here employed along with SOM as an autonomic middleware utility to predict an output based on a given system's inputs data. Multiple regression analysis is generally used to ascertain the relationship between a dependent variable or criterion and a set of independent variables or predictors. In other words, the usage of multiple regression involves the discovery of the relationship between the values and then finding an equation that satisfies such relationship. Multiple regression analysis serves two functions. Firstly, to yield an equation that predicts the dependent variable or criterion from the various independent variables or predictors. Secondly, and more importantly, to identify the independent variables set by controlling the dependant variables. In our model, multiple regression analysis is developed and implemented as an OGSA compliant grid service [144, 145]. Many applications of the multiple regression analysis technique can be found in the public domain [142, 143, 146, 147].

The mathematical model of multiple regression analysis is shown in the following equations:

$$Y_{pi}=\beta_0+ \beta_1x_{1i}+ \beta_2x_{2i}+....+ \beta_k x_{ki}+E_i \quad .....(7.2)$$

Where: i=1, 2,...., n

Y: is the predicted output

X: is the inputs data

E: is the error

k=number of input parameters

n=number of training data

The above equation can be written as follow:

$$Y_1=\beta_0+ \beta_1x_{11}+ \beta_2x_{21}+....+ \beta_k x_{k1}+E_1 \quad ..... (7.3)$$

$$Y_2=\beta_0+ \beta_1x_{12}+ \beta_2x_{22}+....+ \beta_k x_{k2}+E_2 \quad ..... (7.4)$$

$Y_n=\beta_0+\beta_1x_{1n}+\beta_2x_{2n}+....+\beta_kx_{kn}+E_n$ ..... (7.5)

Y, β, and E can be represented as column vector. These are:

$$Y=\begin{bmatrix}Y_1\\Y_2\\.\\.\\.\\Y_n\end{bmatrix} \qquad \beta=\begin{bmatrix}\beta_1\\\beta_2\\.\\.\\.\\\beta_n\end{bmatrix} \qquad E=\begin{bmatrix}E_1\\E_2\\.\\.\\.\\E_n\end{bmatrix}$$

And X can be represented as

$$X=\begin{bmatrix}1 & x_{11} & x_{21} & x_{31} & \dots & x_{k1}\\1 & x_{12} & x_{22} & x_{32} & \dots & x_{k2}\\1 & x_{13} & x_{23} & x_{33} & \dots & x_{k3}\\.\\.\\.\\1 & x_{1n} & x_{2n} & x_{3n} & \dots & x_{kn}\end{bmatrix}$$

Then

Y=Xβ+E ..... (7.6)

To find the matrix formulation for the least-squares estimates for β_0, β_1, β_2, ..., β_k, the system is solved as

Xb=y..... (7.7)

$$(X'X)b = X'y \quad \text{..... (7.8)}$$

Where X' is the X transpose. By multiplying X'Y with the inverse of (X'X), we got:

$$b = (X'X)^{-1}X'y \quad \text{..... (7.9)}$$

Theoretically, to find the least squares estimates for the model parameters, we simply compute

$$\hat{\beta} = b = (X'X)^{-1} X'y \ldots\ldots\ldots (7.10)$$

$\hat{B}$ represents the matrix of coefficients that affect the predicted value. These coefficients are utilised in generating the model for predicting the output. The detail implementation of multiple regression analysis as web service is demonstrated in Sec. 9.4.2.4.

7.3.Framework Design for Self-Management System

Services and infrastructures are the core units of grid computing environment. Such resources are provided by the resource providers in accordance to a contract between the providers and the broker (middleware). Such contracts include rights, privilege and authorisations associated to a given deploy resources -- node or grid system. In addition, the consumers[2] access these resources via their broker (middleware services). Hence, such resources can operate as stand alone services or integrated with other services and infrastructures. Therefore, these resources should be described adequately to the consumers to ease their selection. In addition, the broker should describe how resources can be discovered and invoked.

The middleware naming service is extended with semantic support for improved service discovery. Hence, service meta description has been developed beyond the coverage of the WSDL, in that, it covers resources environment, dependencies, methods, interfaces, required infrastructure, access, Service Level of Agreement (SLA), and other information. The Assembly Services and Infrastructures Framework (ASIF) model consists of three main actors, as shown in the use case diagram of Figure 7.3, which are:

- **Resources provider:** provides the system with the required resources. The resources in this case represent services or infrastructures. Commercial, research, scientific, business, and utilities are examples of services. While super

[2] The term consumer refers to users, applications, and middleware services like fault tolerance, load distributed, and autonomic computing services.

computers, huge distributed database, communication services, and others are examples of the infrastructures.

- **Consumer:** represents users, applications, middleware services, or others. The consumer is responsible for generating the requests that indicate the need for using the resources.
- **Resources container:** contains the deployed resources by the providers. Moreover, it is responsible for generating the response to the consumers' requests.

Different autonomic capabilities are adopted in this framework when performing service discovery, matchmaking and binding (assembly), namely: self-regulation, self-organising, and self-configuration.

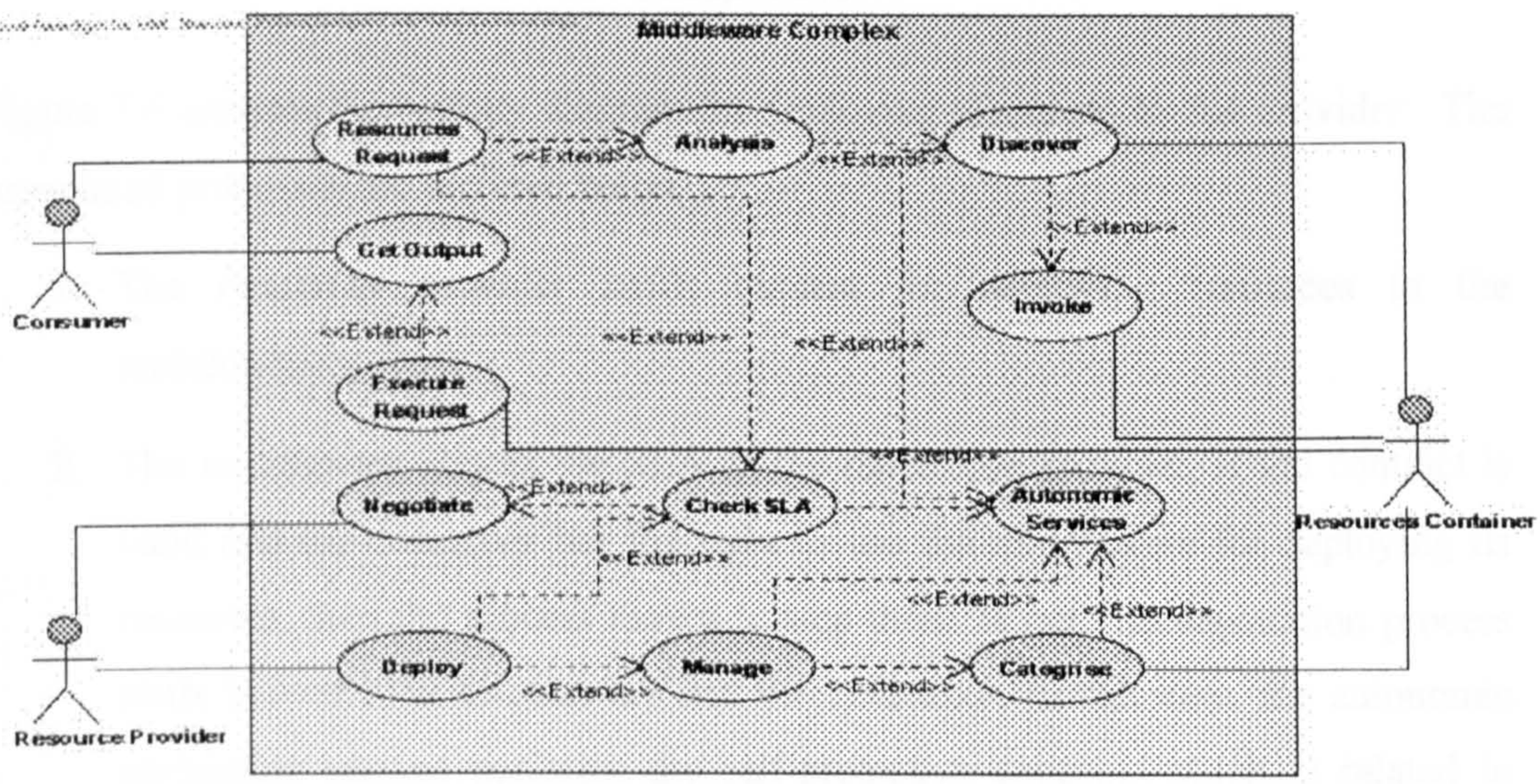


Figure 7.3: UML Use Case Diagram for ASIF

7.3.1. Resources Deployment Process

The sequence diagrams for deploying resources are shown in Figures 7.4 and 7.5. Figure 7.4 demonstrates the business sequence diagram for the interaction between provider and middleware complex, while Figure 7.5 demonstrates the detailed sequence diagram for the same scenario.

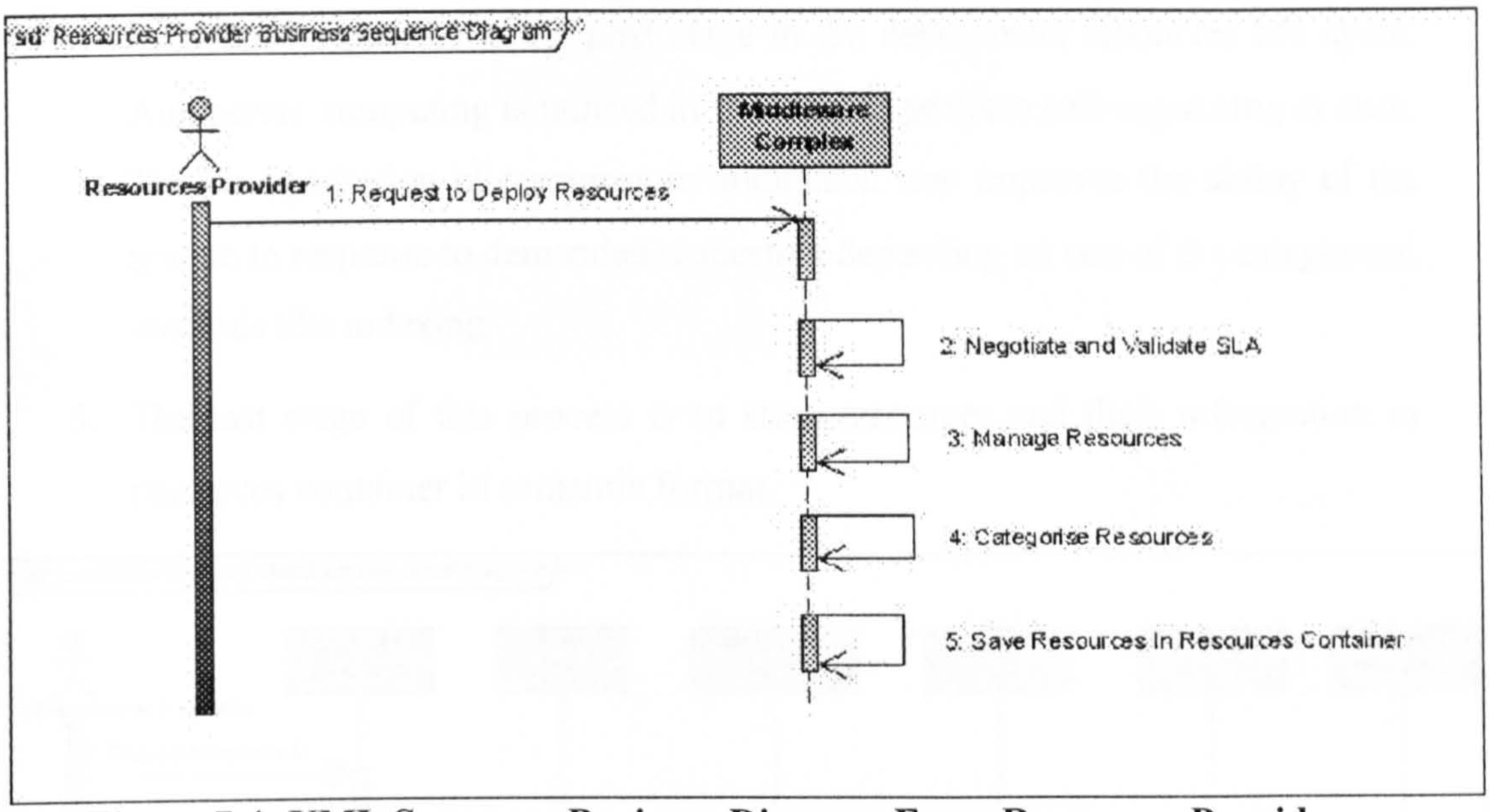


Figure 7.4: UML Sequence Business Diagram From Resources Provider Prospective

Figure 7.6 exhibits the activity diagram for deploying resources by the provider. The associated processes are outlined below:

1. The resources provider sends request for deploying resources to the middleware system.

2. The middleware checks the SLA of the requesting provider. If the contract is valid and the consumer has the right to use the middleware for deploying its resources, then the process moves to step three. If not, the negotiation process starts between the middleware and the consumer. In this case, the autonomic computing service performs the self-protective concept, which is related to checking SLA and implementing negotiation process.

3. Autonomic computing service manages the deployment of resources according to their characteristics, types, functions and SLA. Such management process enhances the ability of the system to control, monitor and use the resources. Moreover, such process is essential to mange the injection of new resources with the existing. In this case, the middleware adopts the autonomic computing capability from Serviceware layer in order to perform the reconfiguration process.

4. Categorise resources is the next stage in the deployment resources life cycle. Autonomic computing is utilised in this case to perform self-organising system. The Categorisation of resources in automated way improves the ability of the system to response to demanded requesting depending on one of the categorised methods like indexing.

5. The last stage of this process is to store resources and their information in resources container in semantic format.

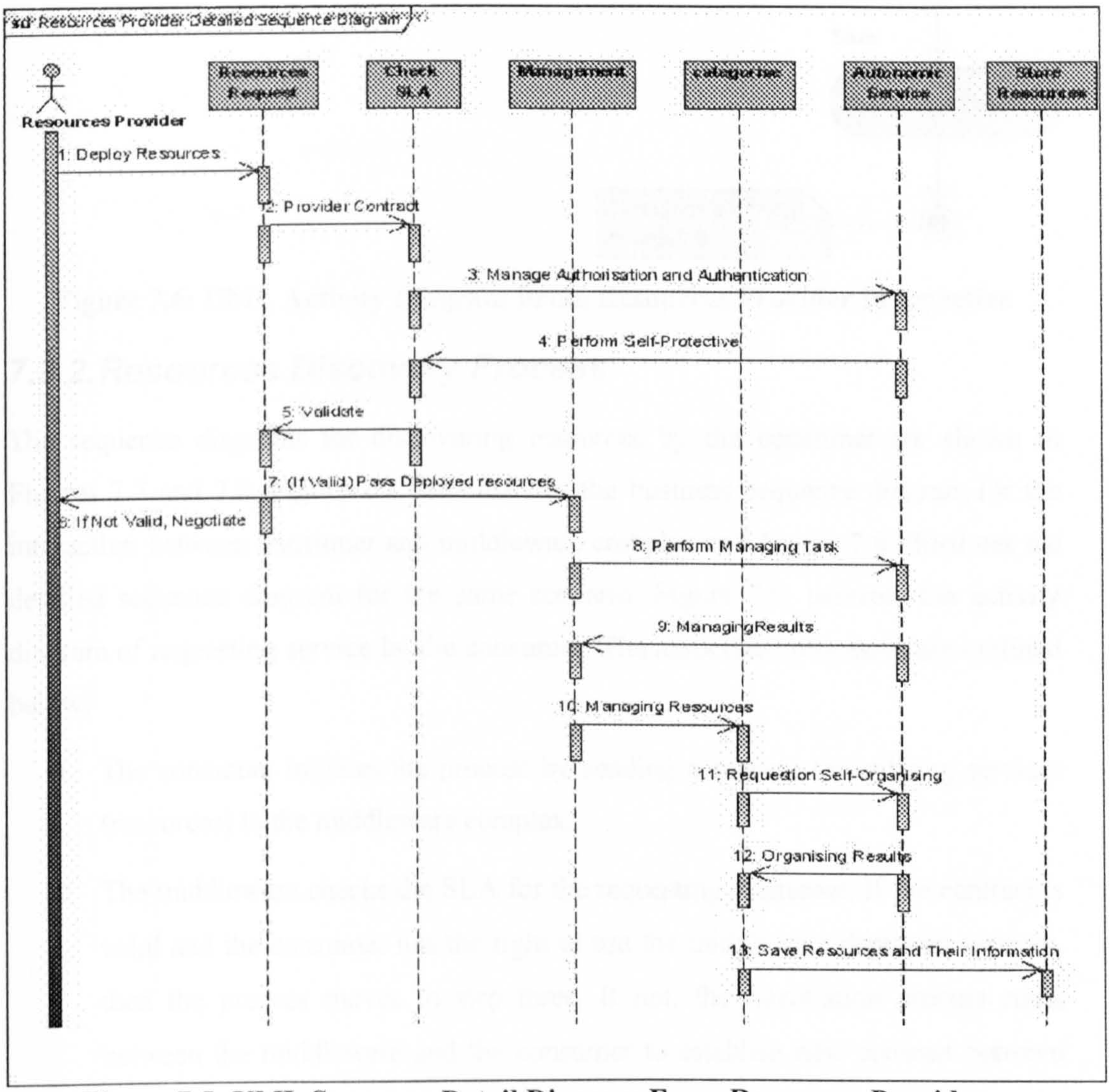


Figure 7.5: UML Sequence Detail Diagram From Resources Provider Prospective

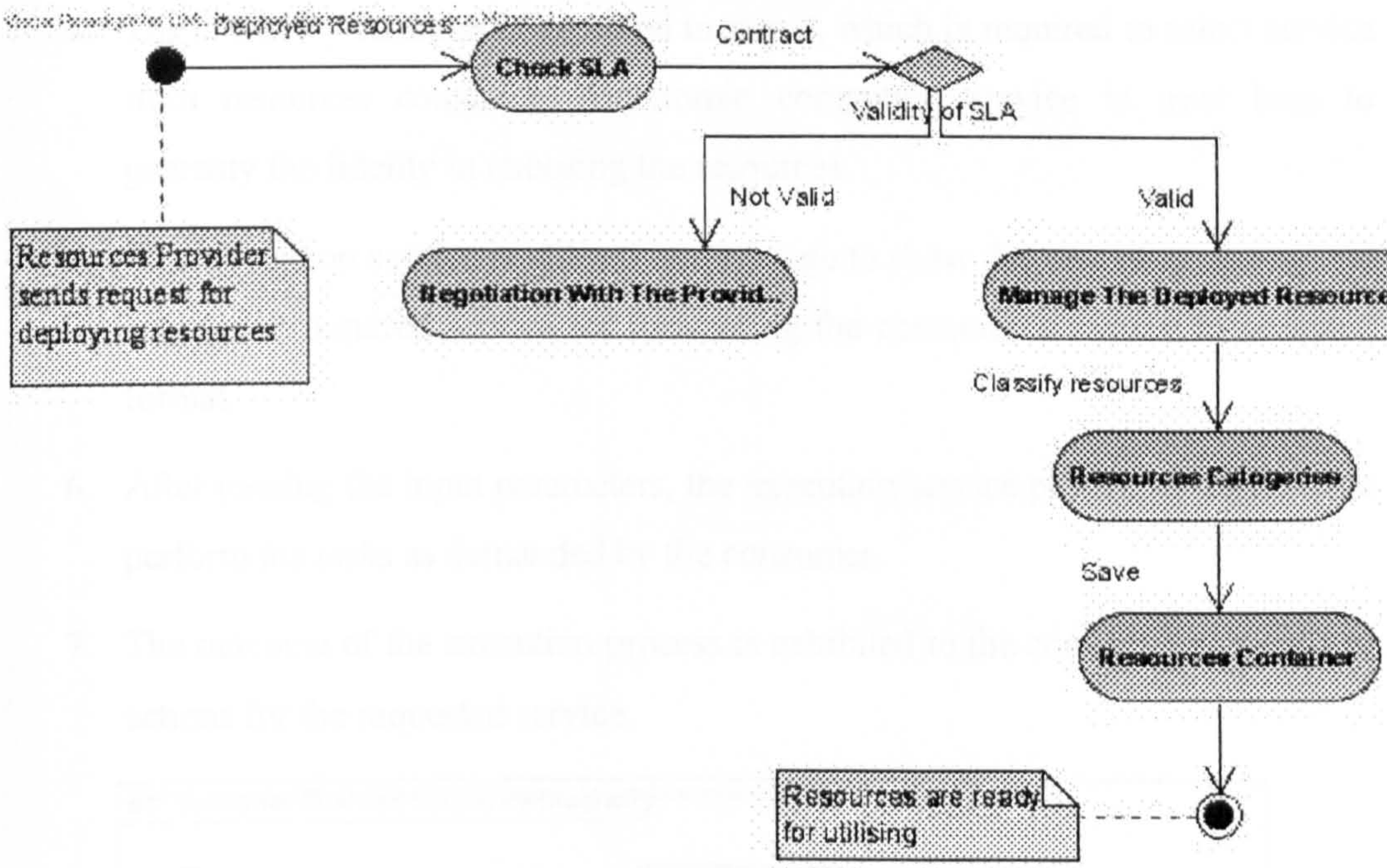


Figure 7.6: UML Activity Diagram From Resources Provider Prospective

7.3.2. Resources Discovery Process

The sequence diagrams for discovering resources by the consumer are shown in Figures 7.7 and 7.8. Figure 7.7 demonstrates the business sequence diagram for the interaction between consumer and middleware complex and Figure 7.8 illustrates the detailed sequence diagram for the same scenario. Figure 7.9, presents the activity diagram of requesting service by the consumer. The associated processes are outlined below:

1. The consumer initiates the process by sending a request for utilising services (resources) to the middleware complex.
2. The middleware checks the SLA for the requesting consumer. If the contract is valid and the consumer has the right to use the middleware deployed services, then the process moves to step three. If not, the negotiation process starts between the middleware and the consumer to establish new contract between the two parties.
3. Middleware analyses the consumer's request to identify the requested services and their dependencies.

4. The discovery function is executed in step 4, which is required to select service from resources container. Autonomic computing service is used here to guaranty the fidelity in choosing the resources.

5. The invocation service is utilised in this stage to show the way of accessing and calling the required service for forwarding the consumer's request in accepted format.

6. After passing the input parameters, the executing service process is triggered to perform the tasks as demanded by the consumer.

7. The outcome of the execution process is exhibited to the consumer as results or actions for the requested service.

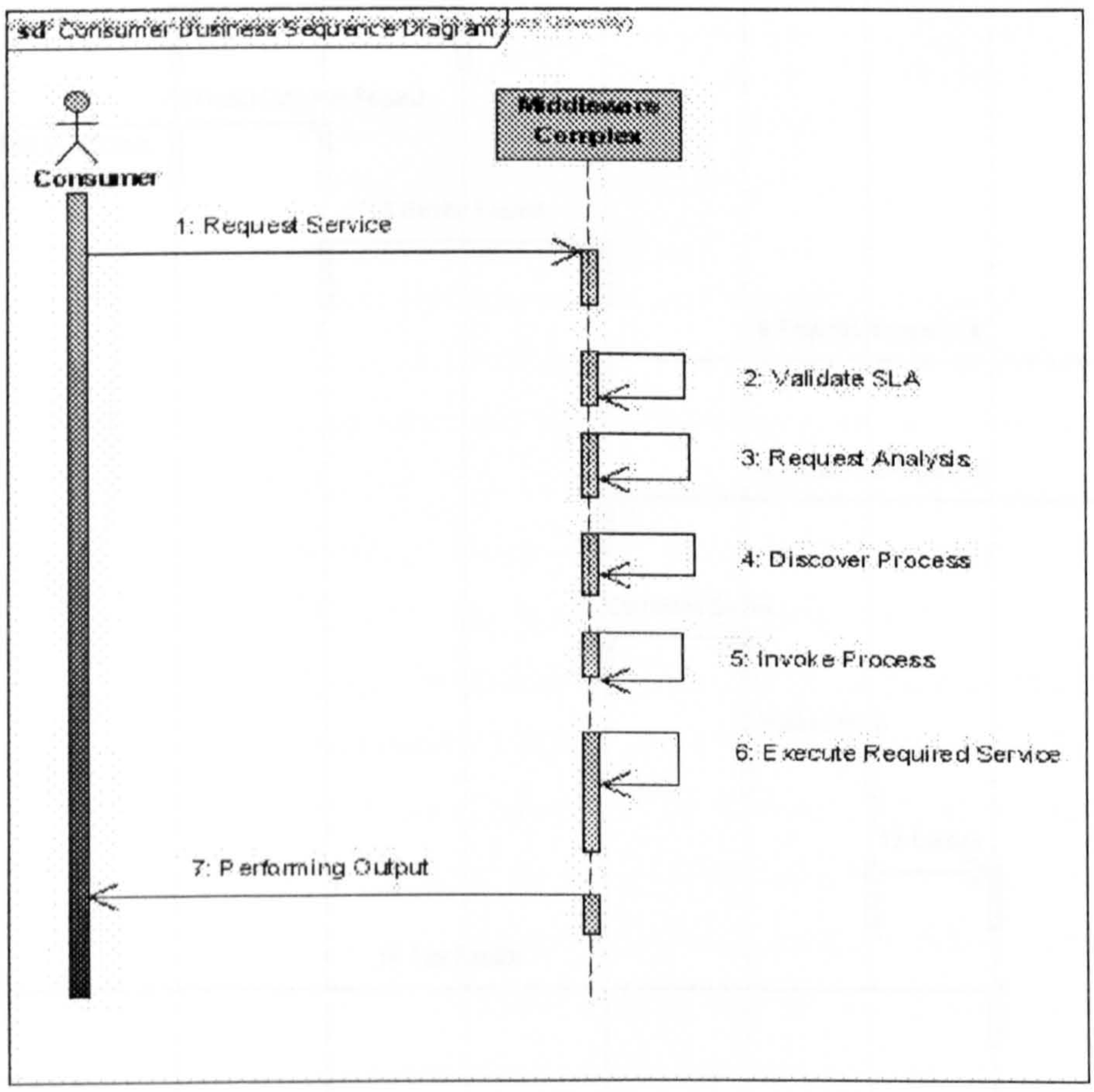


Figure 7.7: UML Sequence Business Diagram From Consumer Prospective

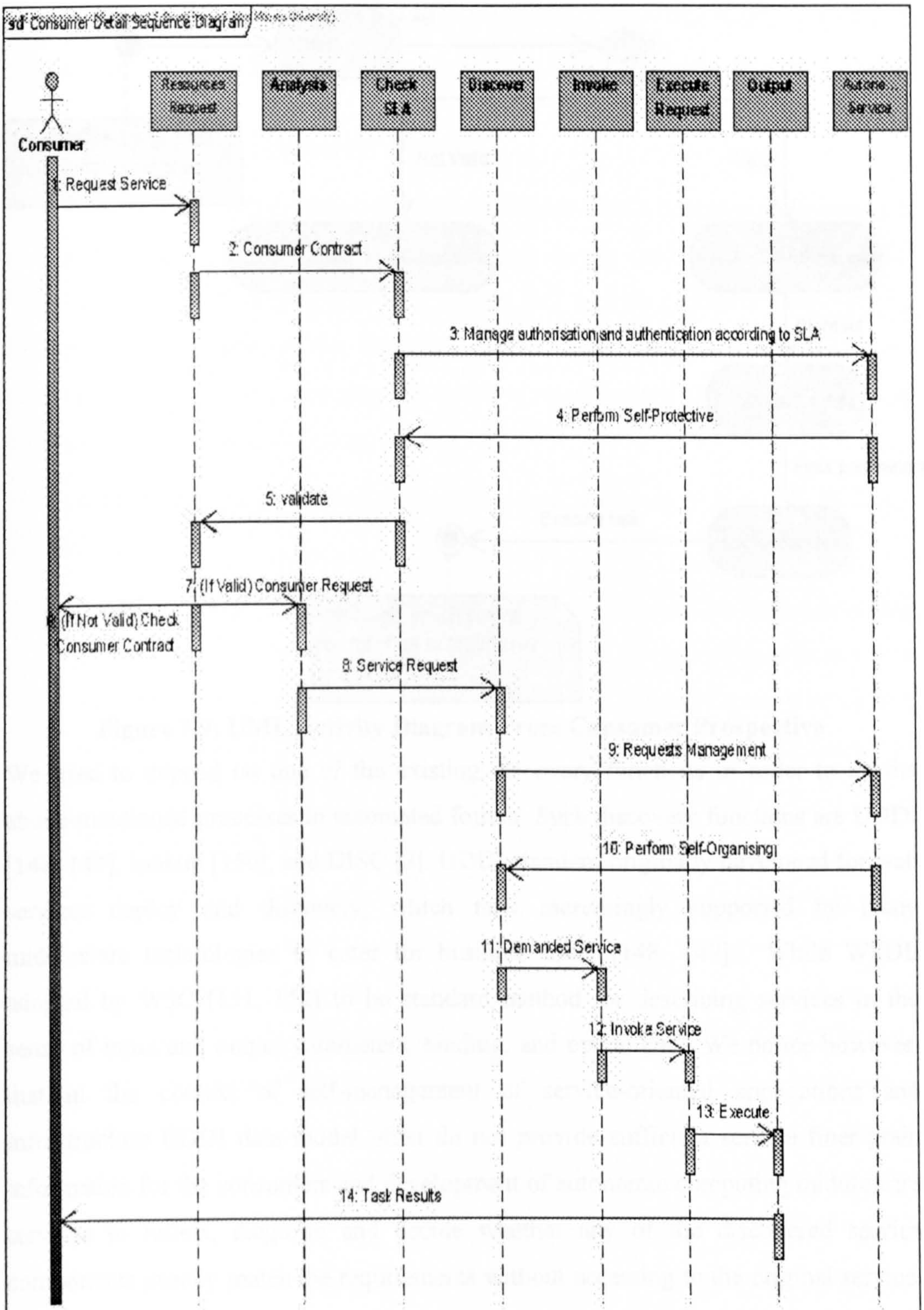


Figure 7.8: UML Sequence Detail Diagram From Consumer Prospective

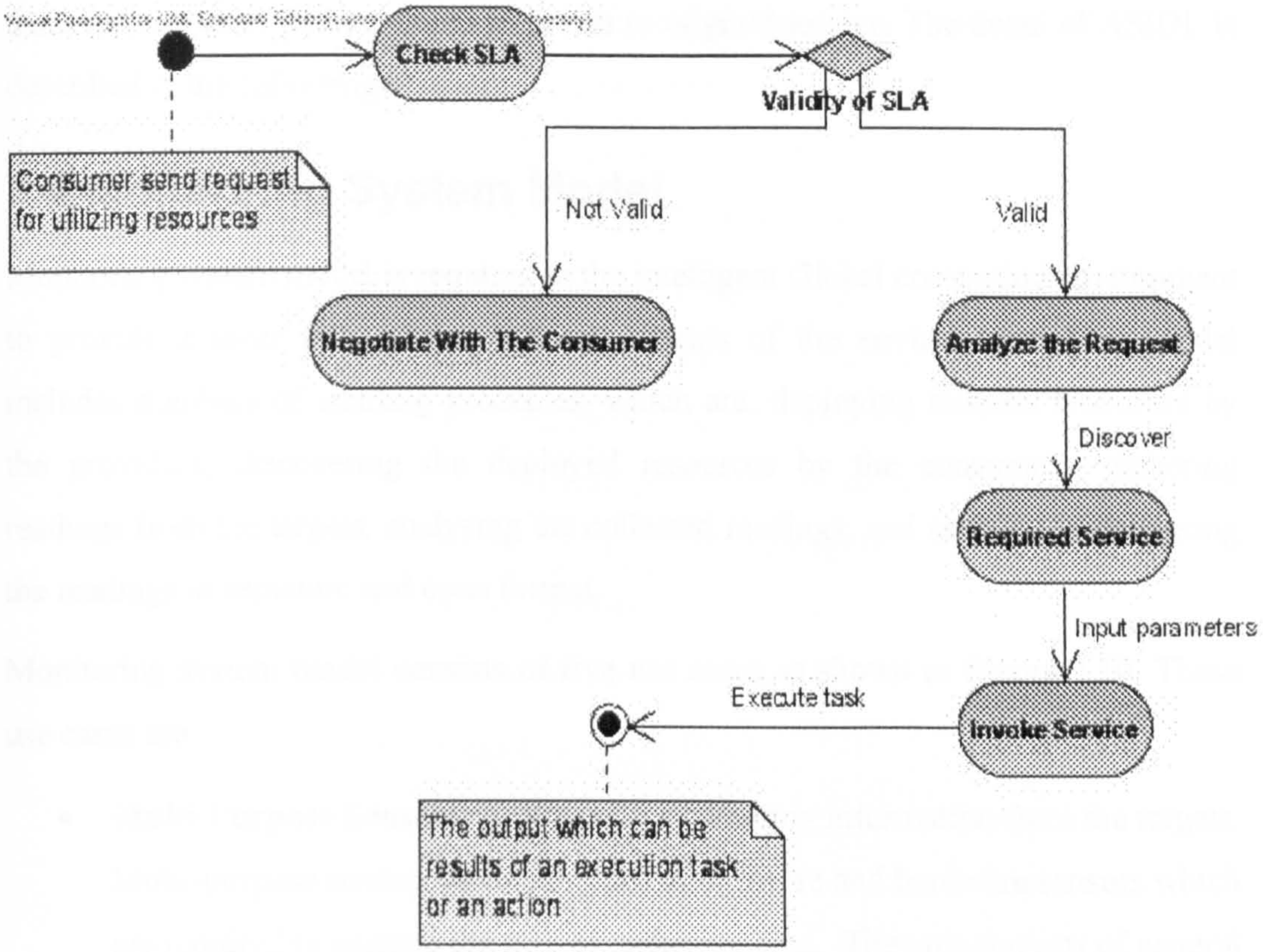


Figure 7.9: UML Activity Diagram From Consumer Prospective

We tried to depend on one of the existing discovery functions in order to do the above-mentioned processes in automated format. Such discovery functions are UDDI [148, 149], lookup [150], and DISC [2]. UDDI standard originally developed for web services deploy and discovery, which now increasingly supported by many middleware technologies to cater for business users [148, 149]. While WSDL adopted by W3C [151, 152] to be standard method for describing services in the sense of input and output parameters, binding, and operations. We notice however, that in the context of self-management of service-oriented applications and infrastructure UDDI data model often do not provide sufficient service finer-grain information for the consumers and development of autonomic computing middleware services to reflect, diagnose and decide whether any of the discovered service components exactly match the requirements without accessing to the original service. Therefore, Assembly Service and Infrastructure Description Language (ASIDL) is designed and developed above WSDL to build a fabric for offering significant information regarding the deployed services and infrastructure for the consumers and

discovery services without needs to access to original service. The detail of ASIDL is described in the following chapter.

7.4.Monitoring System Model

Monitoring system model is required in the intelligent Global computing environment to provide a layer for auditing the components of the environment. This model includes numbers of auditing processes, which are: deploying monitor resources by the providers, discovering the deployed resources by the consumers, gathering readings from the targets, analysing the collected readings, and saving and delivering the readings in semantic and open format.

Monitoring system model consists of five use cases as shown in Figure 7.10. These use cases are:

- **Multi-Purpose Sensors:** is utilised for gathering information from the targets. Multi-purpose sensors cover all types of software and hardware sensors which are required to execute the task of gathering data. The unit consists of general sensors, e-health sensors, connected-home machine sensors, and PlanetLab sensors. Furthermore, this unit is responsible for injecting the sensors in the targets in order to start the process of collecting readings
- **Sensor Container:** includes the deployed monitoring resources from sensors, actuators and analysis services. This container is employed for saving information regarding the deployed resources. Therefore, it represents the layer that is responsible for achieving the fidelity, reliability, availability and QoS in selecting resources. It is corporate with the autonomic computing capabilities in order to discover the most appropriate resources for the requested message. Sensor and Actuation Framework is adopted in this system to deploy, discover and invoke monitoring resources in semantic way.
- **Logger:** provides a container for storing sensors' readings in a standard and semantic format based on the use of XML.

- **Analyser and Actuator:** presents a way for analysing readings. Moreover, it may propose new sensors to be injected in case of unexpected behaviour from the consumer according to the collected readings. Information regarding the proposed sensors is forwarded to the sensor schedule service.
- **Sensor Schedule Service:** receives requests for gathering data from the consumers or monitoring requestors. The sensor schedule service generates a profile for each consumer depending on the information that is emergence from the analyser and actuator unit or from the requester. This profile consists of information regarding the demanded monitoring resources, targets, contract and other information. This profile is forwarded to the sensor container in order to search and inject the required sensors.

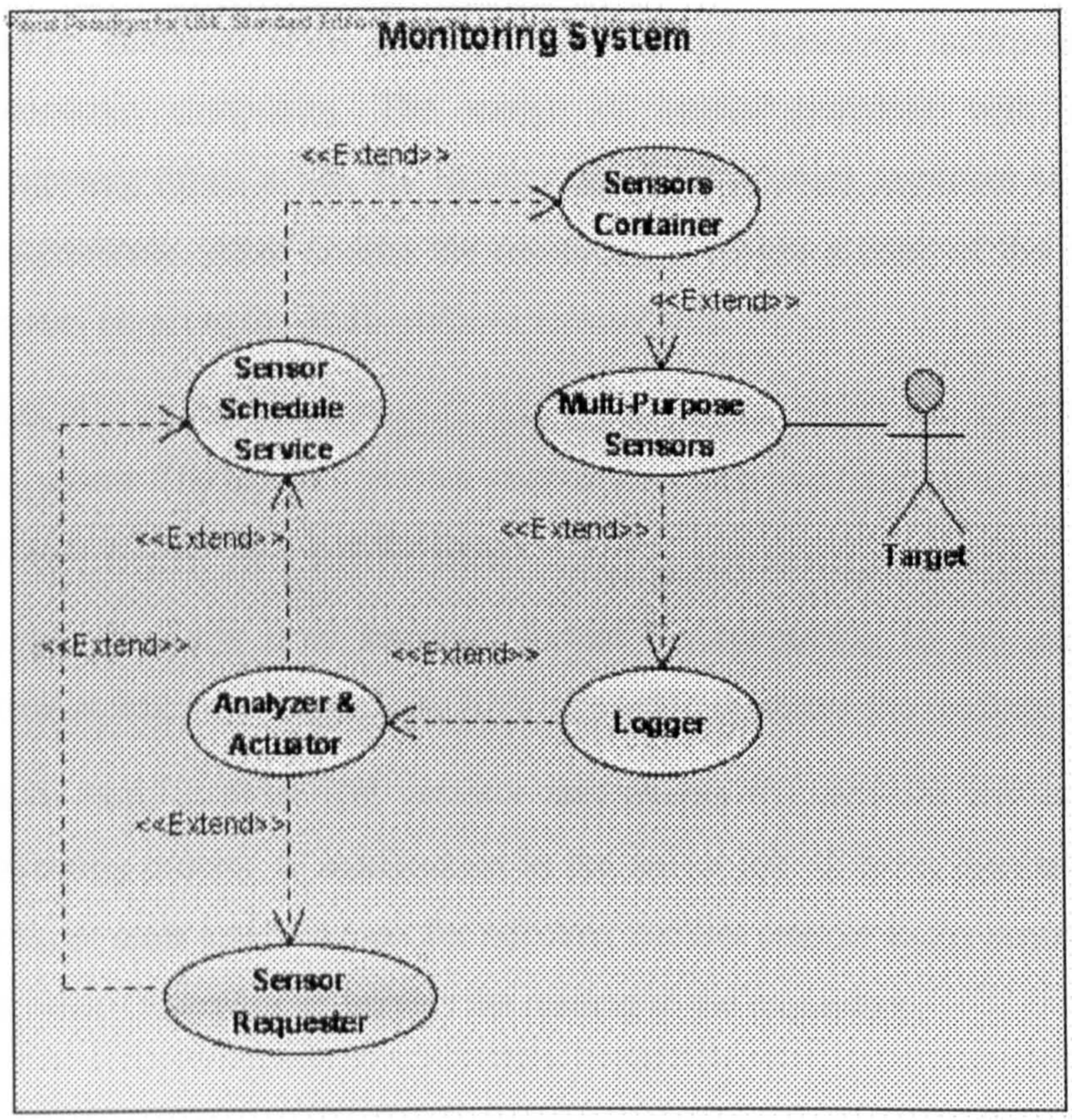


Figure 7.10: Monitoring System-UML Use Case Diagram.

Figures 7.11 and 7.12 show the UML sequence and activity diagrams respectively. The following points describe the scenario of utilising monitor resources based on the proposed monitoring model.

1. The scenario starts by generating request for using monitoring resources to audit specific targets by monitor demander. Monitor Session Description Language (MSDL) is used to pass this information from the requester to the monitoring system in open standard format.

2. The received request is analysed by sensor schedule service to deduce the required information that assists in selecting the required resources, targets, duration of gathering data and application. This information is transferred through a message to sensor container inside SAF.

3. Sensor container starts seek for the required resources incorporation with autonomic computing. The merge of sensor container with autonomic computing offers a way for locating the most relevant resources to consumer requirements. After finding the resources, it moves a copy of the discovered sensors object to the multi-purpose sensors unit.

4. This unit injects these sensors inside the targets, which are specified by the requesters. The injected sensors starts collect data and return it to multi-purpose sensors unit for saving in the logger.

5. Logger converts it to semantic format for submitting to the analyser and actuator unit. In our approach, XML is used to represent the readings.

6. In its turn, analyser and actuator unit checks the data for any error in the collecting process or malfunction in the sensor job. If there, then analyser and actuator units sends request for injecting another sensors to the sensor schedule services. Otherwise, it delivers the reading to the requesters.

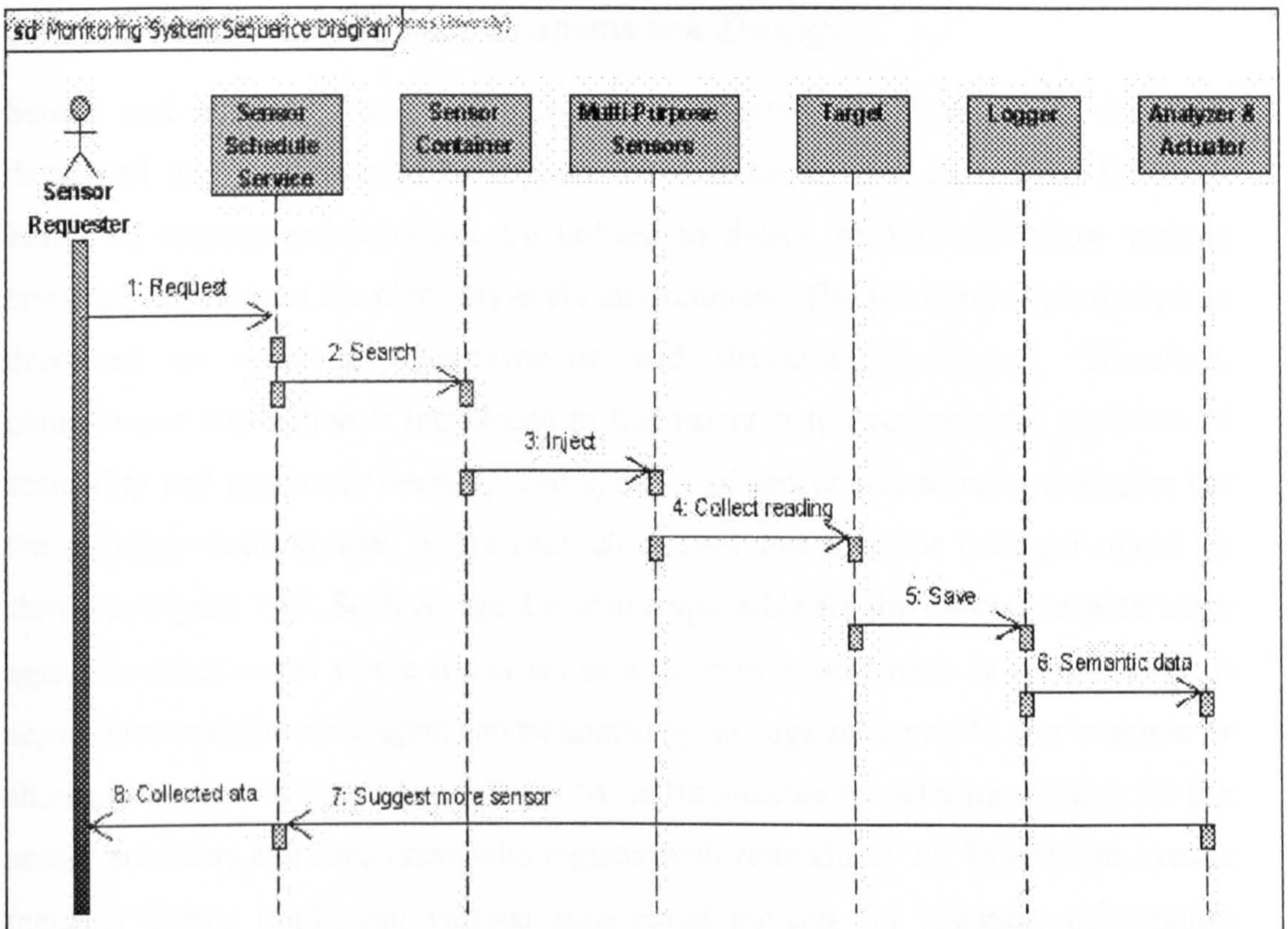


Figure 7.11: Monitoring System-UML Sequence Diagram

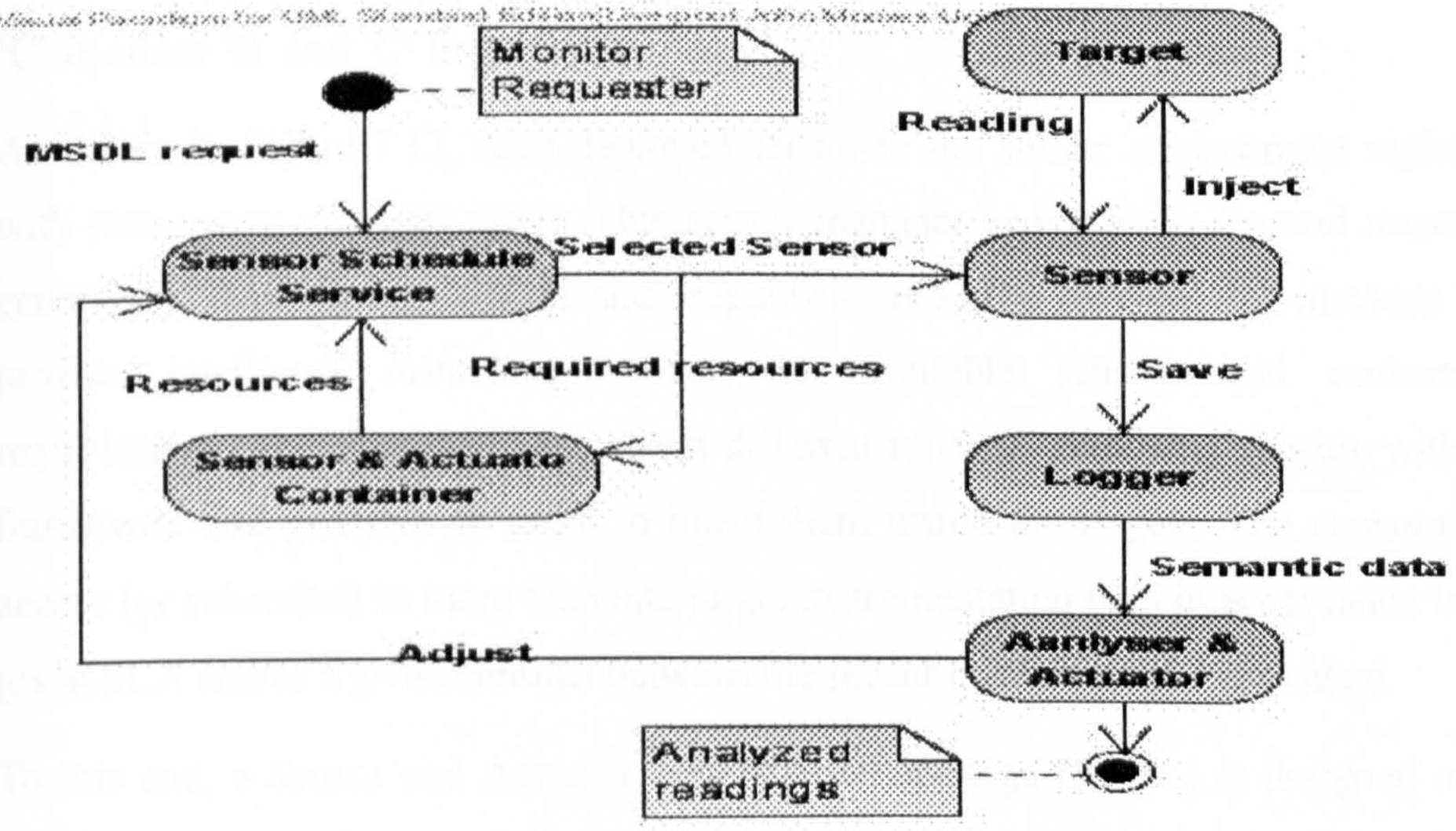


Figure 7.12: Monitoring System-UML Activity Diagram

7.4.1. Sensor & Actuator Framework Design

Sensor and actuator framework (SAF) for planetary-scale system is designed, developed and implemented to support the monitoring and controlling facilities. Range of sensors and actuators are utilised to detect, record, and adjust various systems' activities in the planetary-scale environment. The planetary-scale system is described as scalable, heterogeneous and decentralised system. Therefore, clouds/zones abstraction is introduced in this research to overcome the problems of scalability and massively decentralised systems of sensor and actuator networks for the planetary-scale system, where each cloud/zone has a *sensor manager agent,* as shown in Figure 7.13. Such an agent is also responsible for the interaction with other agents in other zones where it can act as a gateway sensor node to its cloud, as in sensor networks the zone agent can be hosted by an edge sensor node. For example as shown in Figure 7.13, an agent in zone 'A' is responsible for offering services for the sensor providers and consumers who register with zone/cloud 'A'. In addition, sensor manager agents implement various zone-based policies for instance; information exchange, access control and self-healing monitoring and actuation. Moreover, agent 'A' is in charge of taking to other agents in other zones, for instance agents 'B' and 'C' in zones 'B' and 'C' respectively.

As shown in Figure 7.13, each deployed sensor in the sensor environment register with one sensor manager agent. This sensor manager provides on-demand sensors generation, deployment, lookup and/or publish subscribe services for instance to provide intelligent matching between the available sensors and consumer requirements. The consumers can select different types of deploying sensors with a framework and generate an order to inject them inside the targets. The consumers access (or subscribe) to more than one target instrumentation data in accordance to a given SLA and/or a given contract between the sensor consumers and providers.

To this end, a Sensor and Actuator Description Language (SADL) is designed and developed to provide an open standard description mark-up language for lightweight access to deploy sensor and actuators (effectors) to meet the required model in any given zone, which has been described in the following chapters.

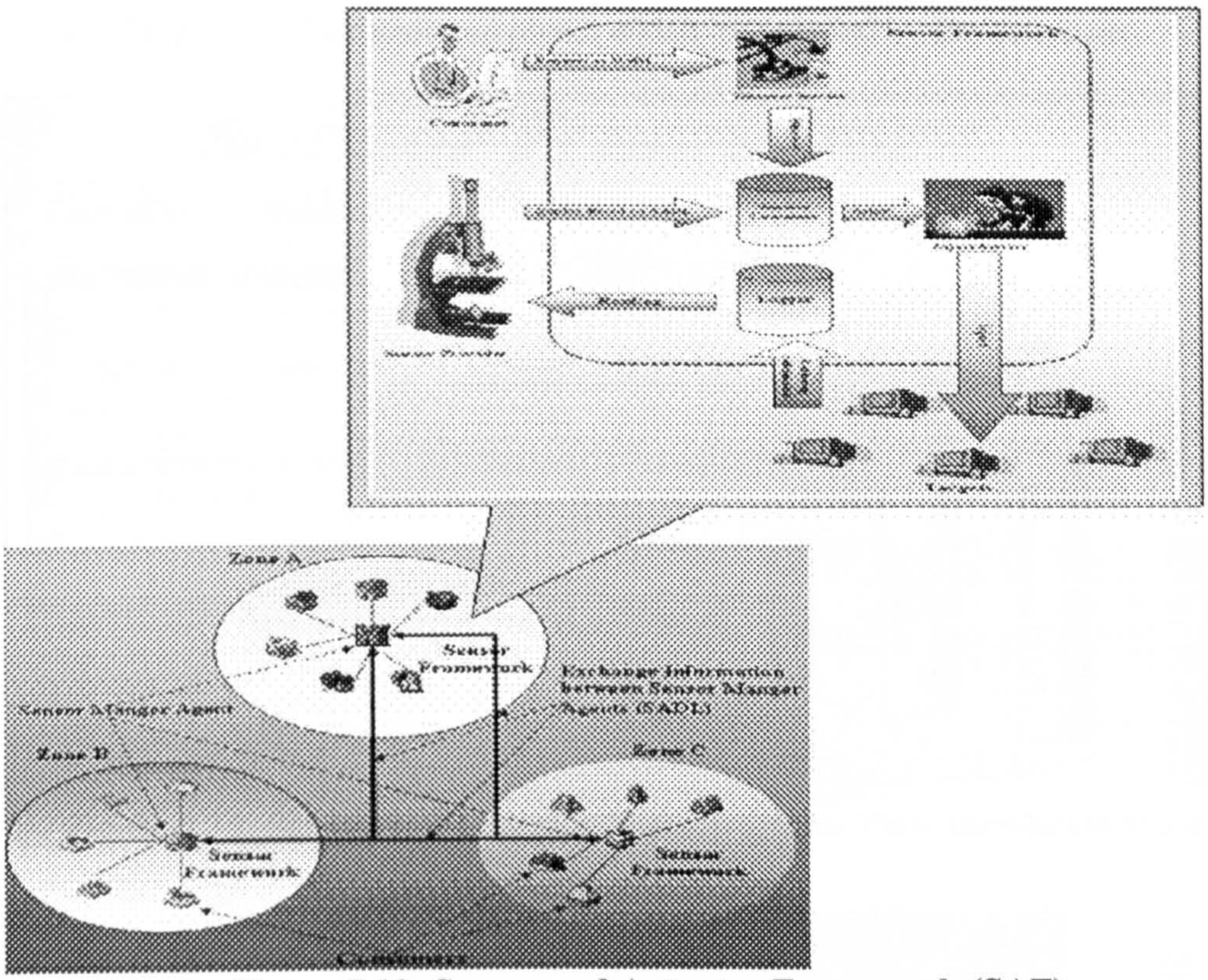


Figure 7.13: Sensor and Actuator Framework (SAF)

7.4.1.1. Sensor & Actuator Framework Scenario

The SAF contains four major actors, as shown in the block diagram in Figure 7.14 and UML use case diagram in Figure 7.15, namely:

- **Sensor and actuator services provider:** deploys and exposes a specific sensor and it's metamodel via the framework in order to be used by consumers.
- **Sensor and actuator container:** is responsible for storing and managing the monitoring resources information, which is deployed by the sensor provider in this framework.
- **Consumers (requester):** generates the request for utilising monitoring resources, which are required to be injected in the targets. The consumer can ask for more than one sensor at a time to be injected in more than one target. The consumer in this case can be sensor schedule service from the proposed monitoring model.

- **Targets:** is the monitored object which is specified by the consumer.

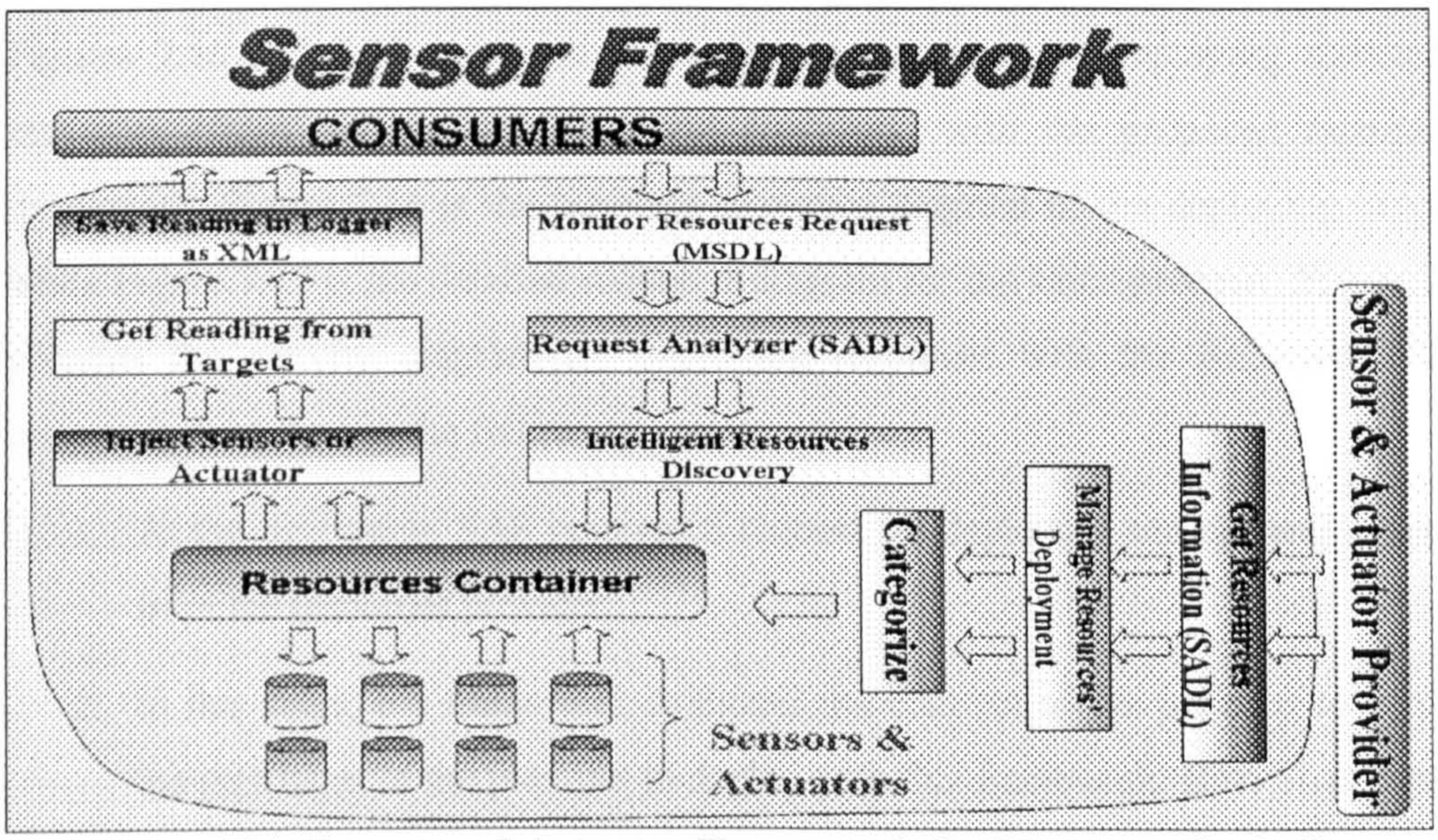


Figure 7.14: Sensor and Actuator Framework for Planetary-Scale System

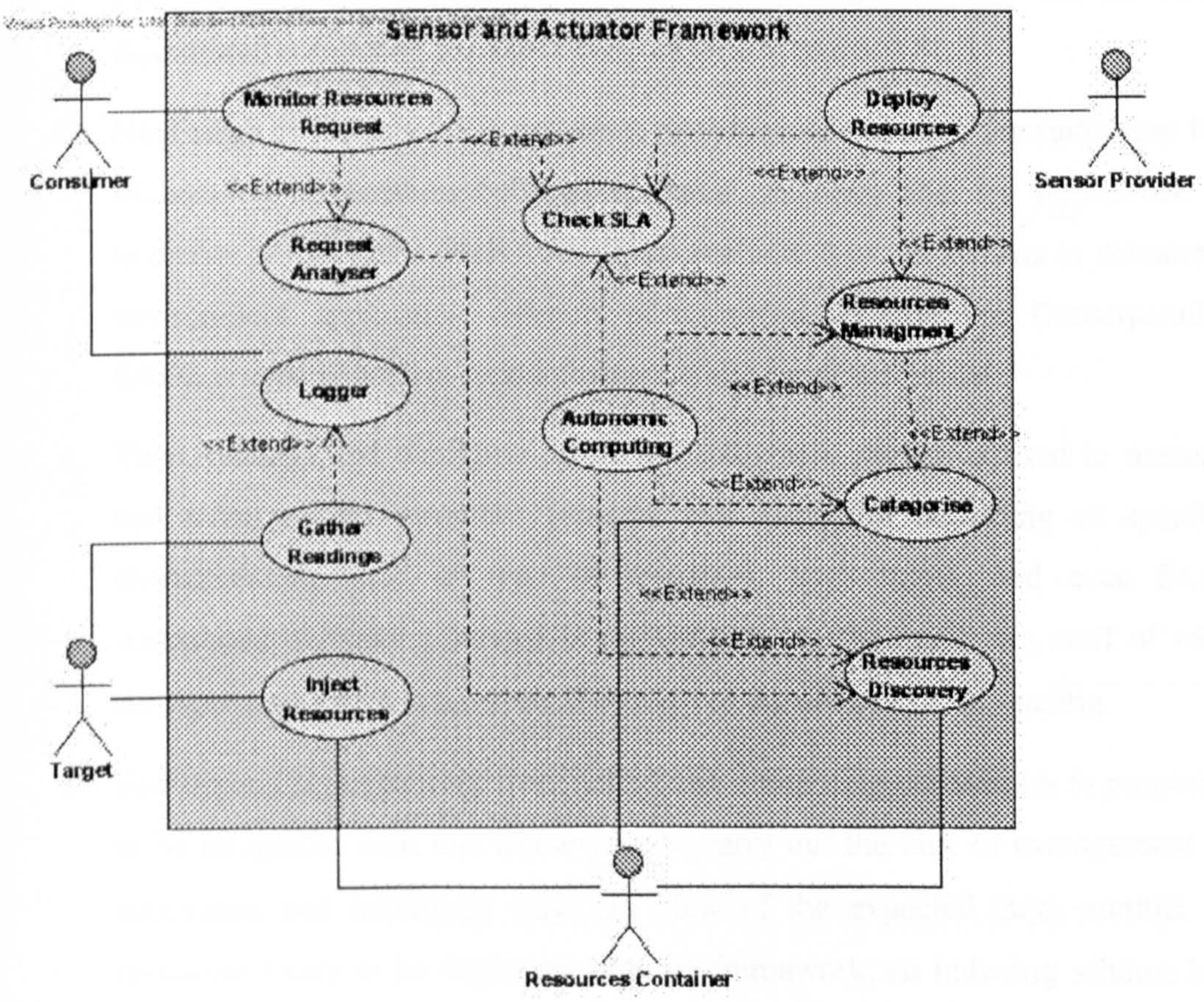


Figure 7.15: Use Case Diagram for Sensor and Actuator Framework

7.4.1.2. Sensors and Actuators Deploying Process

Figures 7.16 and 7.17 represent the sequence diagrams for deploying monitoring resources (from sensors, actuators and analysers) with the proposed monitoring framework. Figure 7.16 is the business sequence diagram of the deploying process, while Figure 7.17 is the detailed sequence diagram for the same process. Figure 7.18 presents the activity diagram for deploying resources with the framework. This process is described in the following points:

1. At the beginning, SAF receives a request for deploying sensors and actuators from the resources provider.

2. In this stage, the framework evaluates the contract of the provider to confirm the privileges of the provider to deploy with this framework. If its SLA is valid, then the system moves to the next step, otherwise the negotiation process occurs between the resource providers and framework to get the best deal for both. Autonomic computing service is adopted here to manage SLA.

3. Next stage of the process is gathering resources information. Semantic way for exchanging information between resource providers and the framework is necessity at this stage. Such information includes types of sensors or actuators, environment, application, category of resources, location, etc. Consequently, SADL is used to present such information in a semantic format.

4. Then, manage and configure resource deployment service is used to manage and integrate the resources (sensors and actuators) according to specific characteristics, such as type of resources, applications, and even SLA. Autonomic computing service is suggested to do the intelligent stuff of self-configuration which is the second concept of the autonomic computing.

5. Self-organising capability from autonomic computing capabilities is proposed to be integrated with this framework to carry out the task of management in automated and intelligent ways. In view of the expected large number of resources likely to be deployed with the framework, an indexing scheme has

been suggested to be used here in order to improve the discovery and selection of sensors.

6. After that, the resources are stored in a container to be ready for utilisation by the framework. The sensor and actuator container is distributed over the planetary-scale system in order to balance the load between the containers. A sensor manager agent is responsible for monitoring and controlling of these distributed resources container.

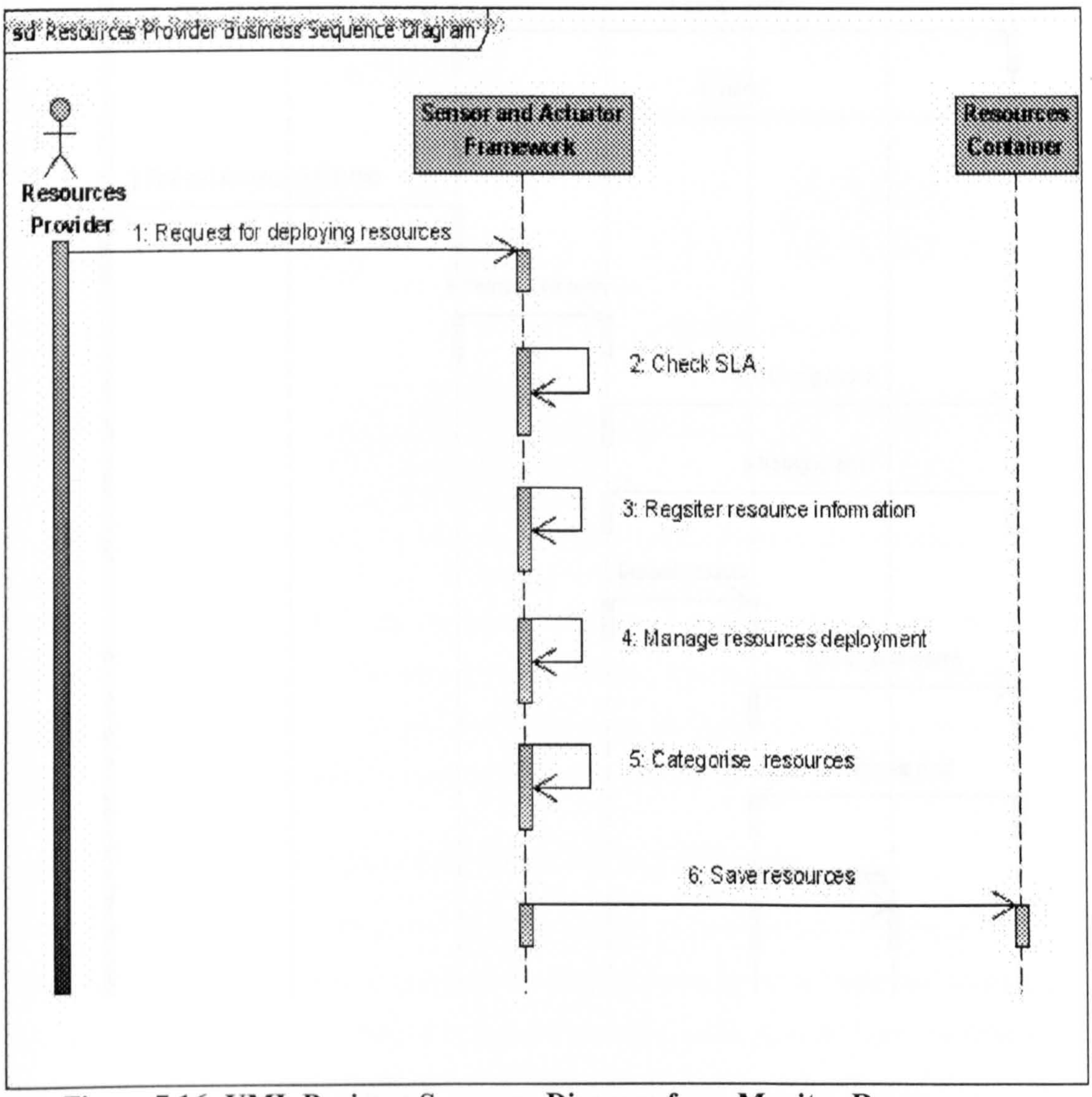


Figure 7.16: UML Business Sequence Diagram from Monitor Resources Provider Prospective

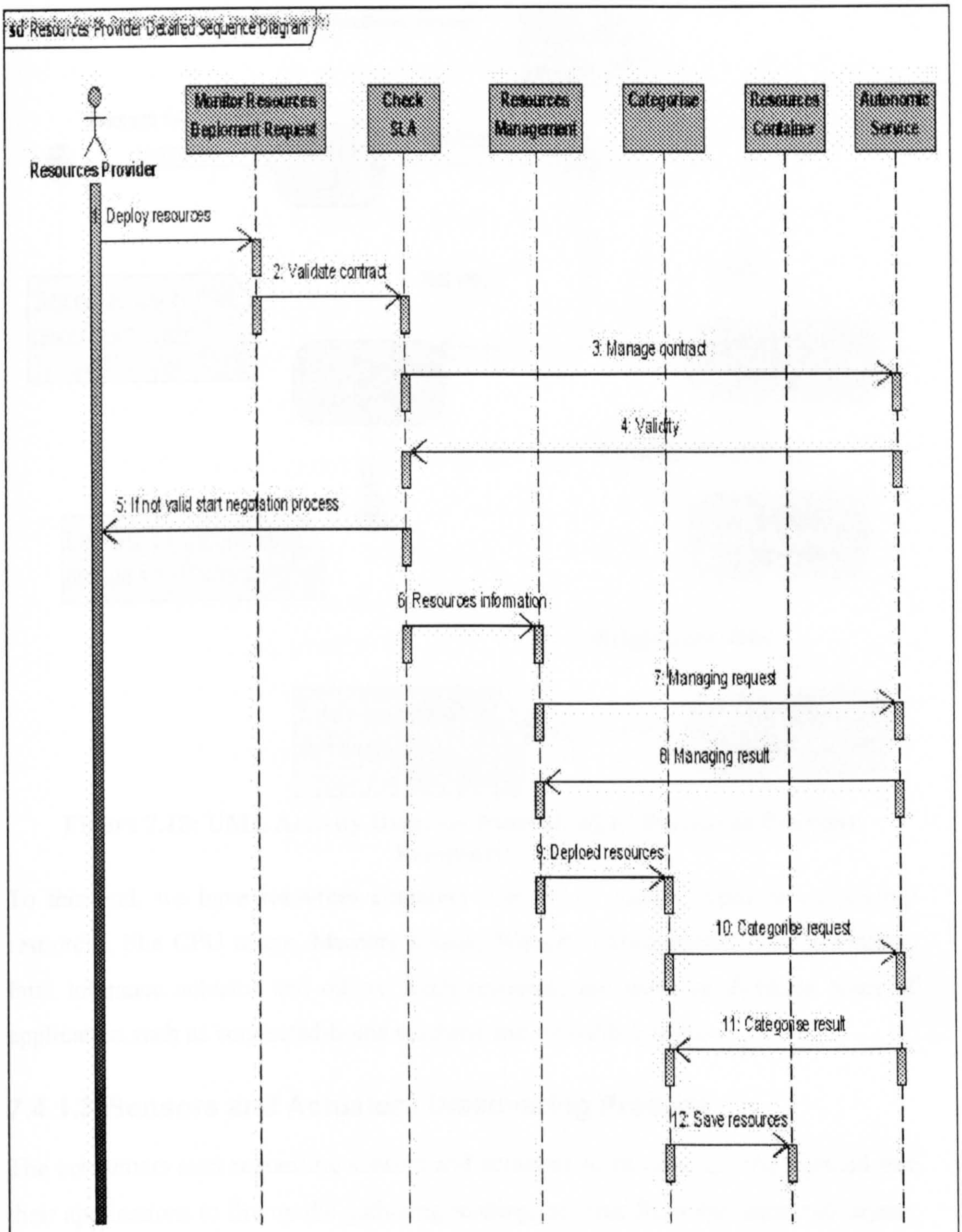


Figure 7.17: UML Detailed Sequence Diagram from Monitor Resources Provider Prospective

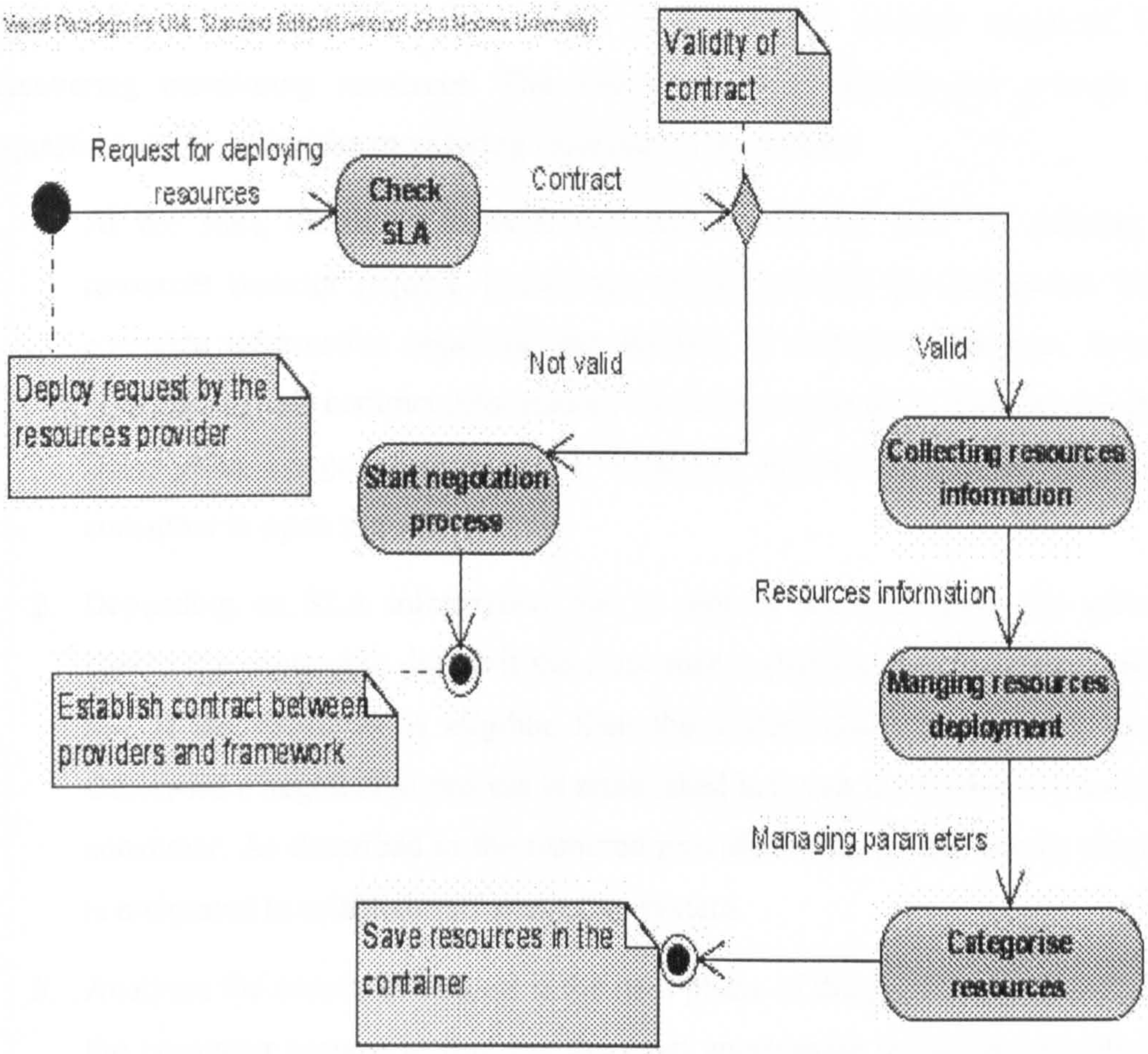


Figure 7.18: UML Activity Diagram from Monitor Resources Provider Prospective

To this end, we have resources container consists of variety types of monitoring resources, like CPU usage, Memory Usage, Weight, Thermometer, Device sensors, fault tolerance actuator and others. Such resources are used for different types of application such as connected-home machine and e-health systems.

7.4.1.3. Sensors and Actuators Discovering Process

The consumers start requesting sensors and actuators to be attached and injected into their applications to fire up the gathering readings process from the requested targets. A target in this case is represented by a host, process, node, device, home or others depending on the application.

Figures 7.19, 7.20 and 7.21 show UML sequence and activity diagrams for discovering monitoring resources. The following steps describe the process of requesting and injecting the monitoring resources in the targets:

1. At the start, consumer initiates connection with the SAF by sending a resources monitor request. Consumers should provide the framework with complete information regarding the duration of collecting readings, targets information, and contract information which represents SLA. Monitor Session Description Language (MSDL) is employed to send a request from the consumer in open standard format.

2. Depending on SLA information that is sent by the consumer, the system checks the contract to decide if the consumer is eligible to utilise resources or not. If the consumer is eligible, then the system shifts to the next stage. Otherwise a negotiation process is established between the framework and the consumer. As described in the resource provider scenario, autonomic service is integrated to establish self-protective system.

3. Analyses the consumer request is the next phase of the process. SAF analyses the consumer request to find out the most appropriate resources suitable for the consumer's requests and contract between the consumer and the framework.

4. Autonomic service starts to perform intelligent searching to identify the well matched resources for the requested mission. Moreover, autonomic service is responsible for managing consumers' demands in a way to give better response, reliability, resources availability and fidelity.

5. After finding the required resources, the system injects them into the targets in order to start collecting measurement data or performing an actuated action.

6. The collected data are stored in a log file inside the logger to be ready for delivering to the consumers in XML format after analysing the reading to be sure from their validly.

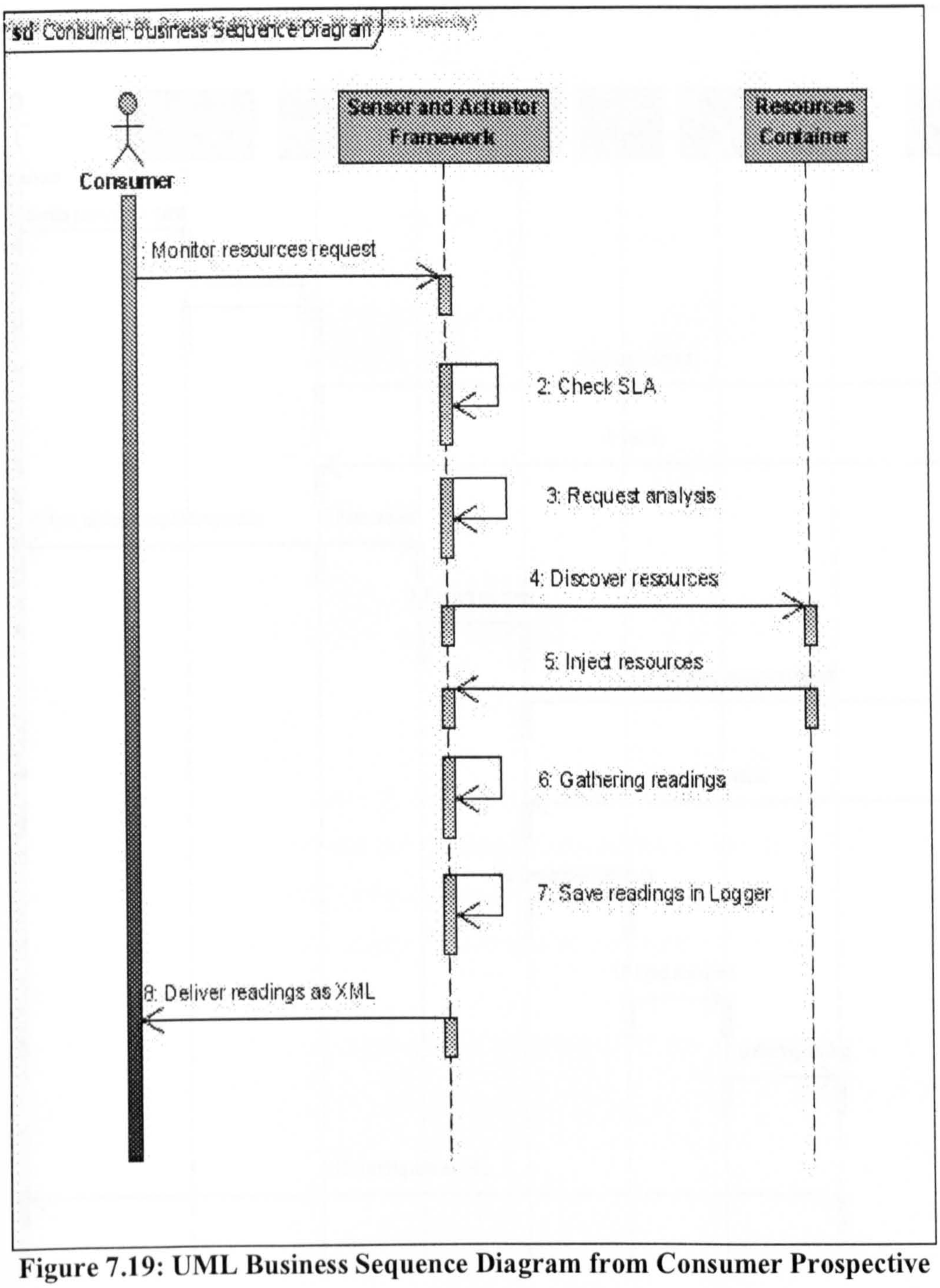


Figure 7.19: UML Business Sequence Diagram from Consumer Prospective

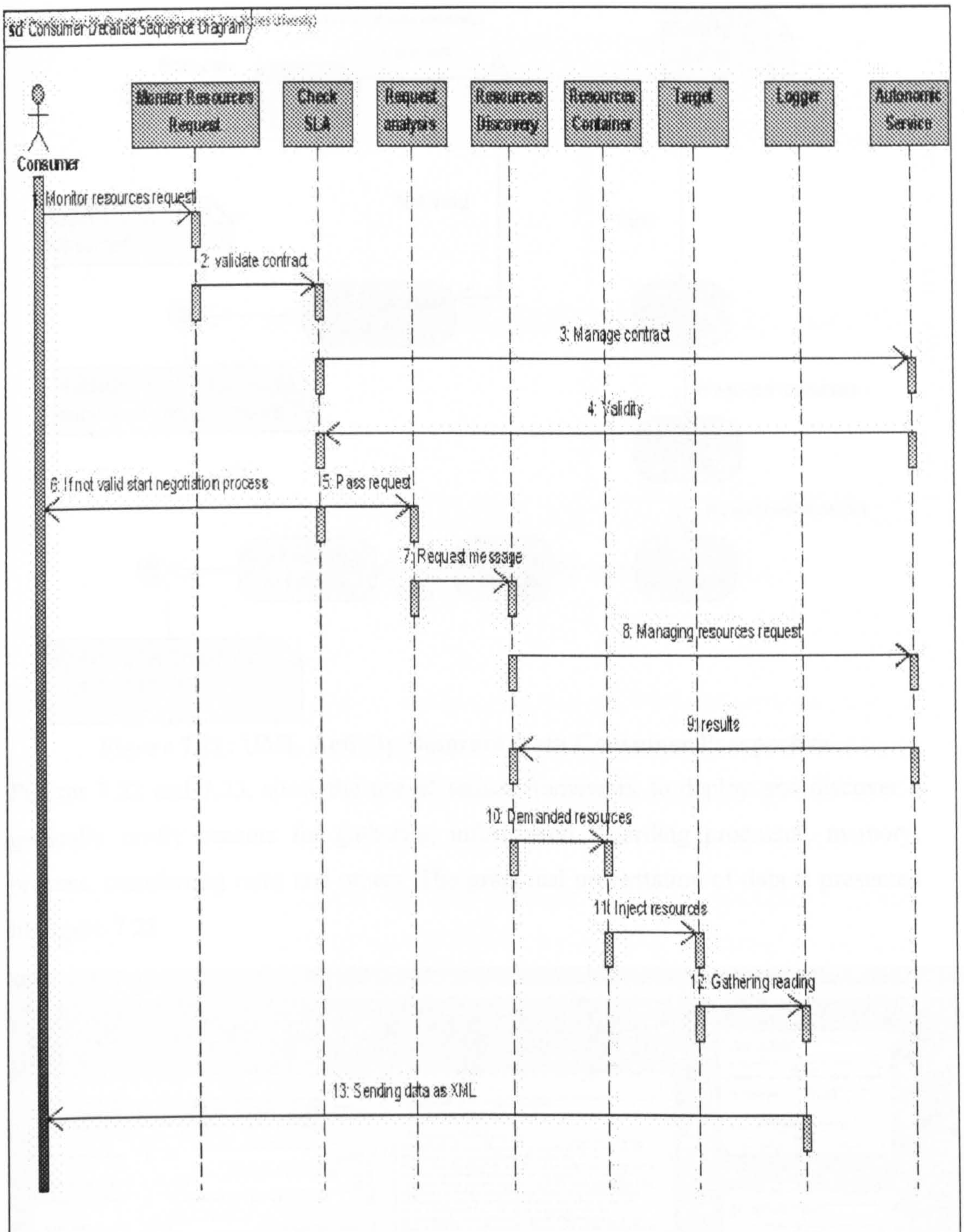


Figure 7.20: UML Detailed Sequence Diagram from Consumer Prospective

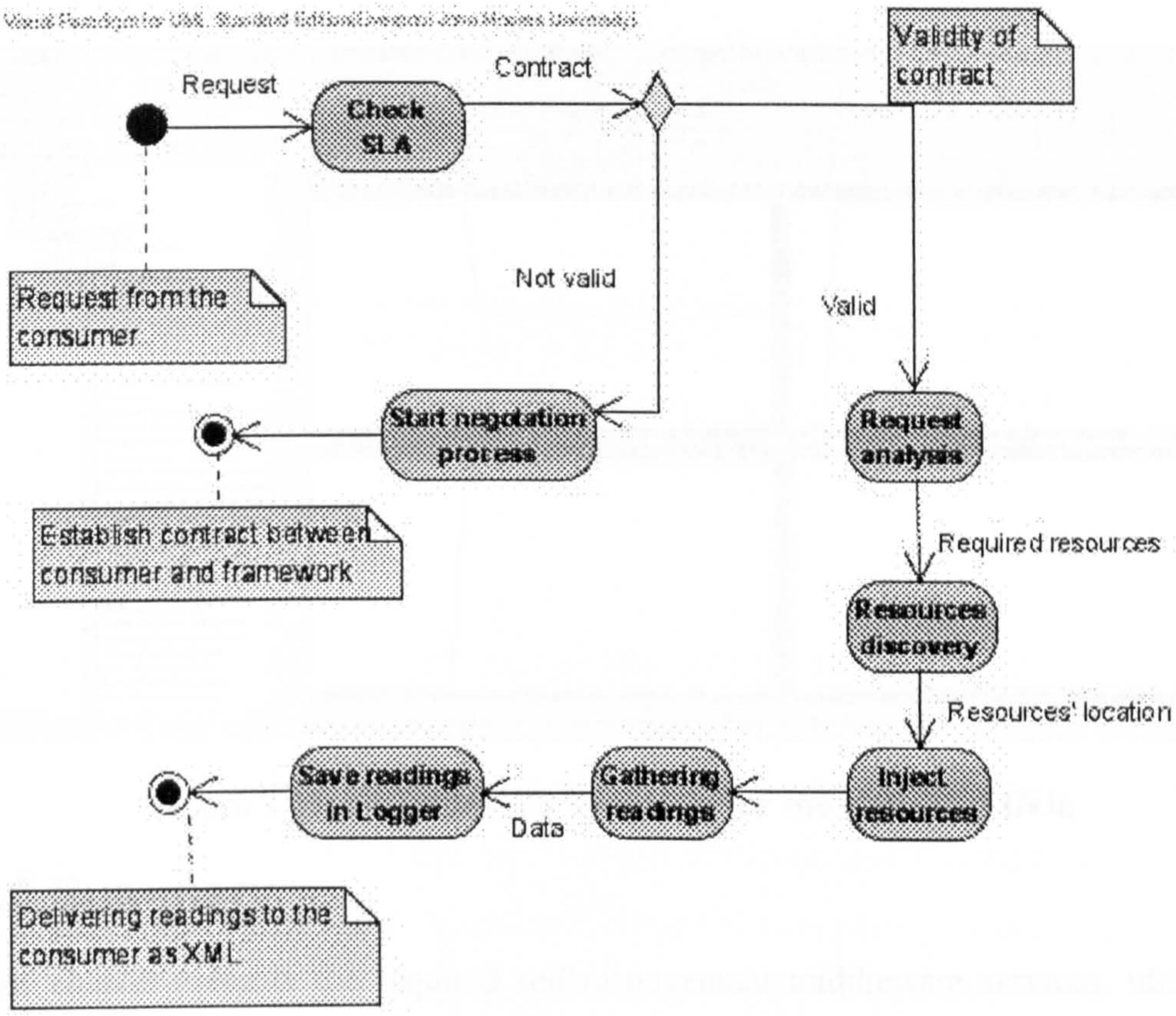


Figure 7.21: UML Activity Diagram from Consumer Prospective

Figures 7.22 and 7.23, show the use of sensor framework to deploy and discover a generally on-fly sensors for gathering information regarding processor, memory, process, transferring rates and others. The graphical presentation of data is presented in Figure 7.23.

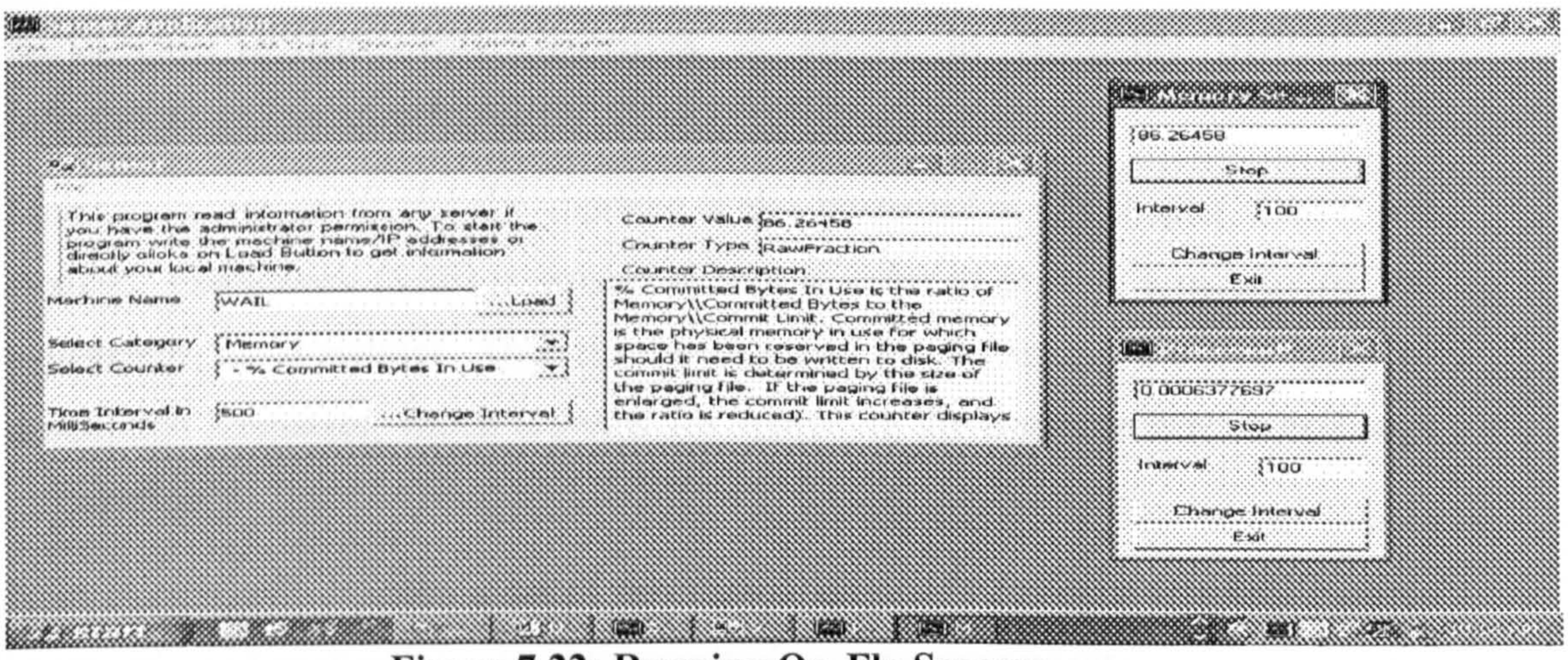


Figure 7.22: Running On-Fly Sensors

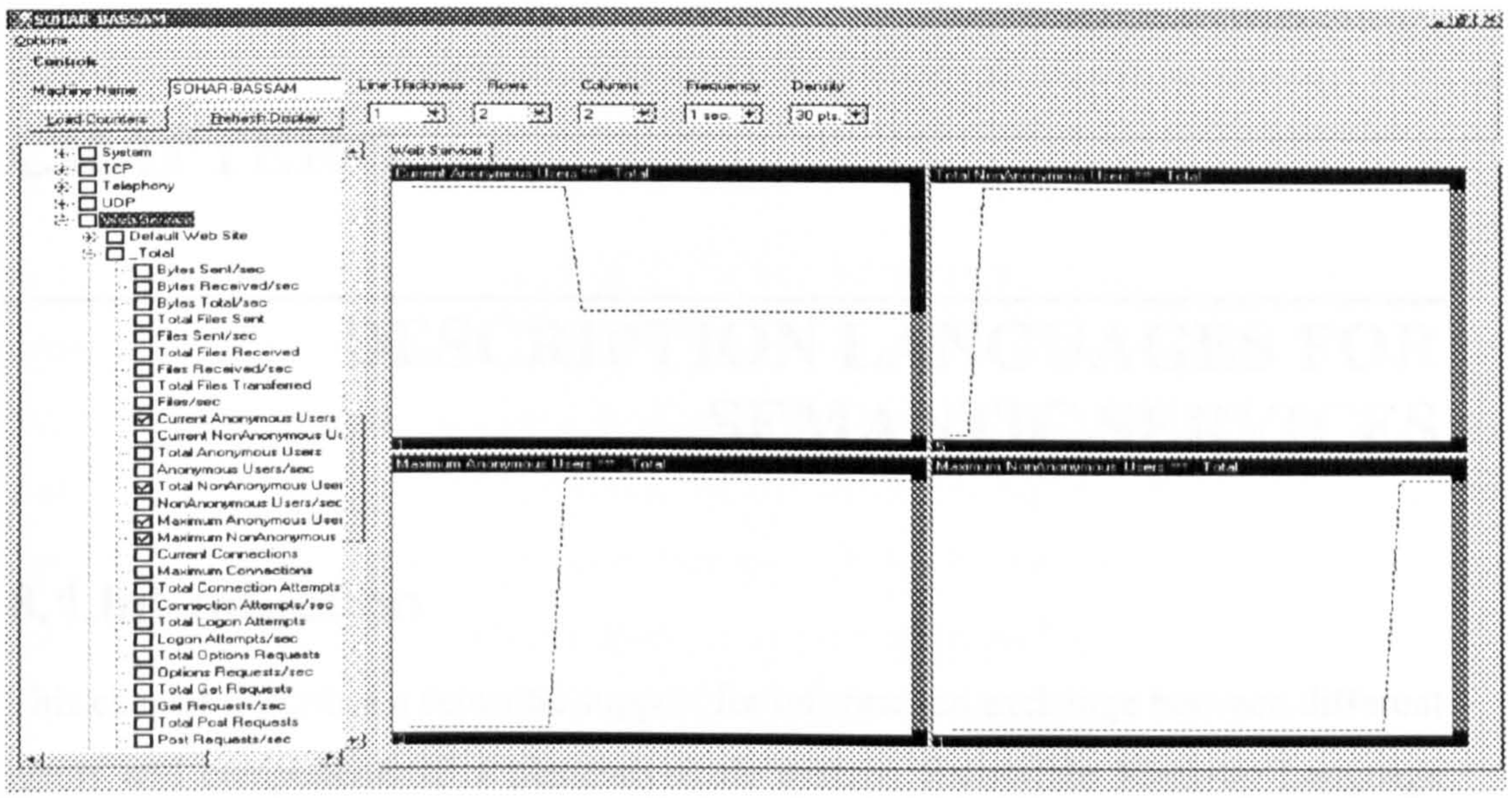


Figure 7.23: Graphical Presentation for the Collected Data

7.5. Summary

This chapter presents the required self-management middleware services, utilities, and frameworks to support autonomic grid computing. A design model of intelligent service is used to illustrate the need and use of a machine learning middleware service to support learning and evolution of operating model of self-managing systems.

Moreover, a framework for deploying, discovering, invoking and managing recourses is adopted in this research to be the fabric for offering resources to the self-management system. This framework is known as Assembly Serices and Infrastructures Framework (ASIF)

Monitoring system for global computing is also presented in this chapter. Sensor and Actuator Framework (SAF) is employed here in order to provide a monitoring and adjustment functionalities. This chapter is ended by discussing the two types of description languages, which are vital for approving the semantic way in exchange information.

Next chapter will describe a set of description languages for open standard runtime autonomic grid application assembly, monitoring and management.

CHAPTER 8

DESCRIPTION LANGUAGES FOR SEMANTIC SERVICES

8.1.Introduction

This chapter describes a semantic support for information exchange between different actors and components of a planetary-scale system. Extensible Mark-up Language (XML) is employed to design and develop a set of open standard description languages namely; (i) Assembly Services and Infrastructures Description Language (ASIDL), (ii) the Sensor and Actuation Description Language (SADL), (iii) Monitor Session Description Language (MSDL). Each of which will be detailed below.

8.2.Assembly Service and Infrastructures Description Language

As presented in the previous chapter, there is a need for an open standard semantic support to facilitate the interoperability between different actors and components of a given planetary-scale system. Current Common Information Model (CIM) [153] and others provide some support. For instance, the CIM common definitions enable vendors and users to exchange semantically wealthy management information between systems throughout the network. However, CIM metamodel does not describe deployed resources from grid services and infrastructures beyond what can be accessed via WSDL and UDDI. This feature is judged essential for planetary-scale self-managing systems. Hence, this work proposed and implemented an Assembly Services and Infrastructures Description Language (ASIDL) that assists the user to select the best match services according to his needs and more importantly provides a rich and accessible metamodels of a given assembly service (application). ASIDL

provides access to three types of metamodel namely; assembly container, services and infrastructures. Each of which will be detailed in the following sections.

8.2.1. Assembly Container Section

Assembly container is in charge of collecting information regarding the deployed resources. The resources are expressed as services and infrastructures. Each unit of the assembly container includes a number of services and infrastructures, which are deployed by one owner. Each service or infrastructure has its own specification, methods, interfaces, resources, environment and SLA. The detail parameters of assembly container are shown in Figure 8.1 and described in Table 8.1. Moreover, an example of using assembly service for deploying number of general services and infrastructures are shown in Figure 8.2.

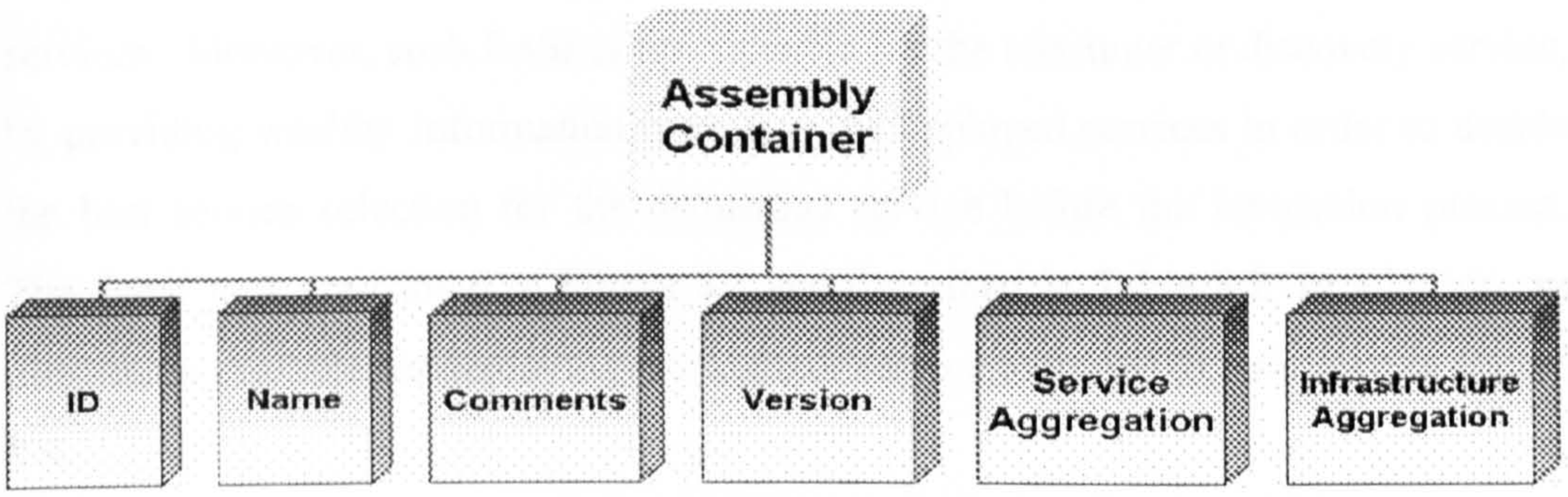


Figure 8.1: Assembly Container Tags

Table 8. 1: Assembly Container's Tags

Elements Name	Description
Assembly_ID	Assembly ID should be unique and generated automatically by the system.
AssemblyName	The name of the assembly.
AssemblyComments	Any comments, which are added by the provider in order to describe the contents of the container.
AssemblyVersion	The version of the assembly. This is valuable for future integration of different versions of assembly containers.
ServiceAggregation	The IDs of the services deployed with this container.
InfrastructureAggregation	The IDs of the infrastructures deployed with this container.

```
<ServiceAssembly SA_ID="1">
        <Name>Research Application Service</Name>
        <Comments>Example of services assembly description
        language</Comments>
        <Version>1</Version>
        <ServiceAggregation>1</ServiceAggregation>
        <ServiceAggregation>2</ServiceAggregation>
        <ServiceAggregation>3</ServiceAggregation>
        <InfrastructureAggregation>I1</InfrastructureAggregation>
        <InfrastructureAggregation>I2</InfrastructureAggregation>
                                </ServiceAssembly>
```

Figure 8.2: Assembly Services

8.2.2. Services Section

This section of the ASIDL offers information concerning with the deploying services. A number of features need to be specified by the service's owner in order to assist the middleware system in managing, categorising, controlling and monitoring the services. Moreover, such features are valuable for the consumer or discovery service, by providing wealthy information regarding the deployed services in order to decide the best service selection for the demanded service before the invocation process. The detail tags are shown in Figure 8.3 and described in Table 8.2. In addition, an illustrative example for deploying a calculator service with ASIDL is shown in Figure 8.4.

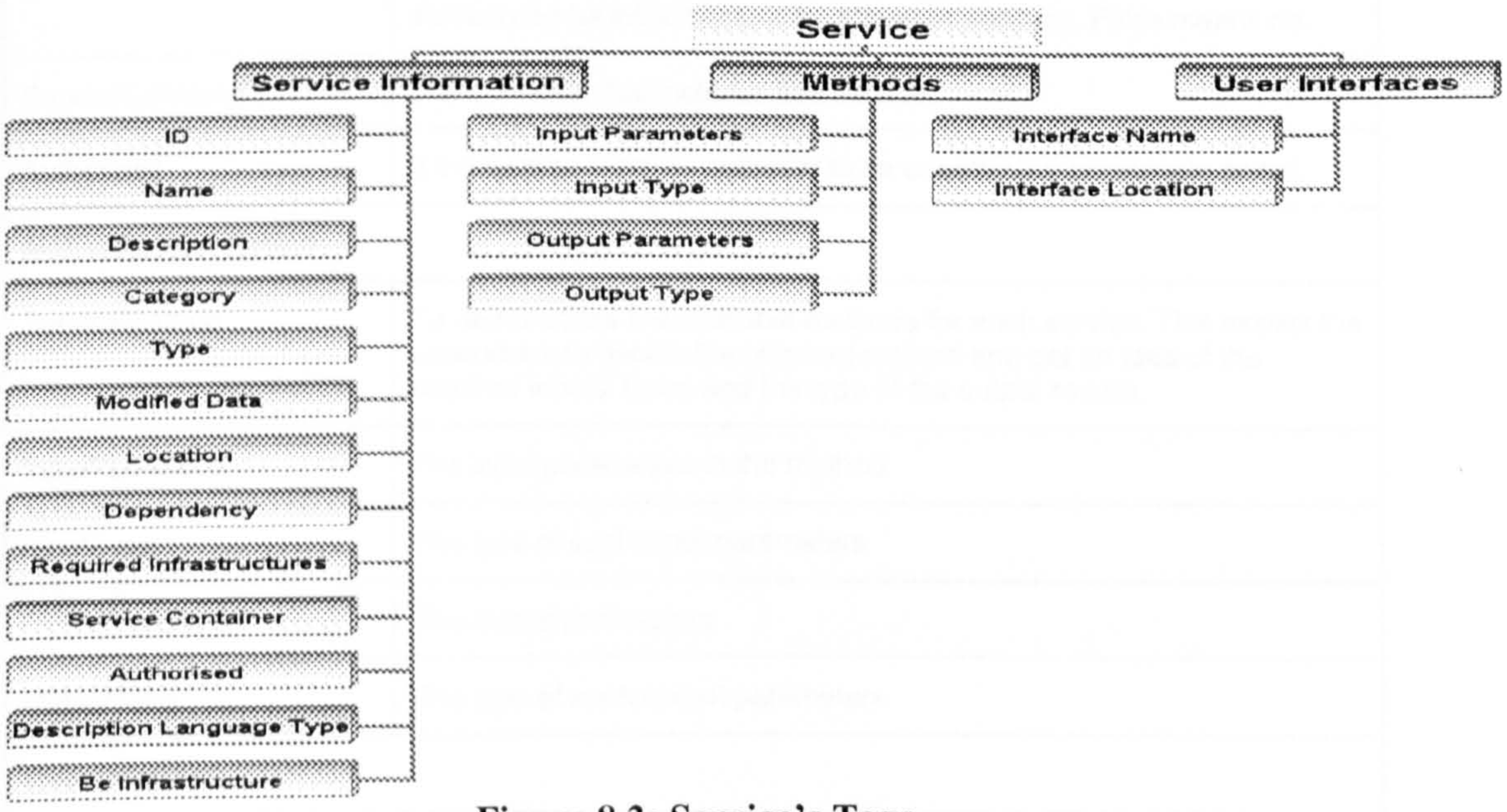


Figure 8.3: Service's Tags

Table 8. 2: Details Service's Tags

Elements Name	Description
Service Information	
ServiceID	ID of the service.
ServiceName	The name of the service.
ServiceDescription	The description of the deployed service.
Description LanguageType	This field describes the type of description language that is used to describe the service, i.e. WSDL.
ServiceCategory	This tag is used to classify each service with specific category. This is useful in services categorised or indexing processes.
ServiceLocation	Location of the service.
ServiceType	To indicate the type of the service. The type can be considered as research, government, commercial, etc....
ModifiedDate	To indicate the last update for the services. This is useful to make the consumer knows which version he/she/it has been utilised.
Dependency	The required objects, DLLs, platforms and other dependencies to make the services implement the process.
RequiredInfrastructure	Describes the required infrastructures to run the service.
BeInfrastructure	This element takes two values, true or false. True means that this service can be infrastructure for the other services. False means no.
ServiceContainer	The container that includes this service.
Authorised	If the service required authority to be used by the consumers or not.
Methods Information	
ServiceMethod	To demonstrate the available methods for each service. This assists the consumers to decide the required method and get an idea of the required inputs' types and the type of the output results.
InputParameter	The input parameters to the method
InputType	The type of each input parameters
OutputParamet	The output parameters
OutputType	The type of each output parameters
User Interface	

InterfaceName	To describe the available user interfaces or user agent which can be used to request the service and get the result.
InterfaceLocation	

```
<Service ServiceID="1">
      <ServiceName>Calculator</ServiceName>
      <ServiceDescription>Do the basic Arithmetic
Operation</ServiceDescription>
      <ServiceCategory>Basic Calculater</ServiceCategory>
      <ServiceType>Research</ServiceType>
      <Container>cmsgris</Container>
      <ServiceLocation>cmsgris:8080/userdoc/Calculator
      </ServiceLocation>
      <Dependency>Antecedent </Dependency>
      <Authorized>True</Authorized>
      <AvailableLanguage>Eng,Fr,Ar</AvailableLanguage>
      <Be_Infrastrucre>True</Be_Infrastrucre>
      <RequiredInfrastructure />I1<RequiredInfrastructure />
      <UserInterface>
            <InterfaceName>Calc.exe</InterfaceName>
            <InterfaceLocation>www.livjm.ac.uk\cmpwomar\
              research</InterfaceLocation>
      </UserInterface>
      <Methods>
            <MethodName>getCalcInterface</MethodName>
            <InputParameters>a, b</InputParameters>
            <InputParameterTypes>Double, Double</InputParameterTypes>
            <OutputParameters>c</OutputParameters>
            <OutputParameterTypes>Double</OutputParameterTypes>
      </Methods>
      <ContractInformation>
            <ContractID>1</ContractID>
            <ServicesContractName>SCCmsgris</ServicesContractName>
            <ServicesContractLease>1/1/2004</ServicesContractLease>
      </ContractInformation>
</Service>
```

Figure 8.4: Services Example

8.2.3. Infrastructures Section

As described in Chapter 7, infrastructure section provides a fabric for performing the computation, communications, databases, software instrumentations, basic arithmetic operations and other services. The detail tags of infrastructures are shown in Figure 8.5 and described in Table 8.3. An example of using infrastructure description language for deploying an addition process with ASIDL is shown in Figure 8.6.

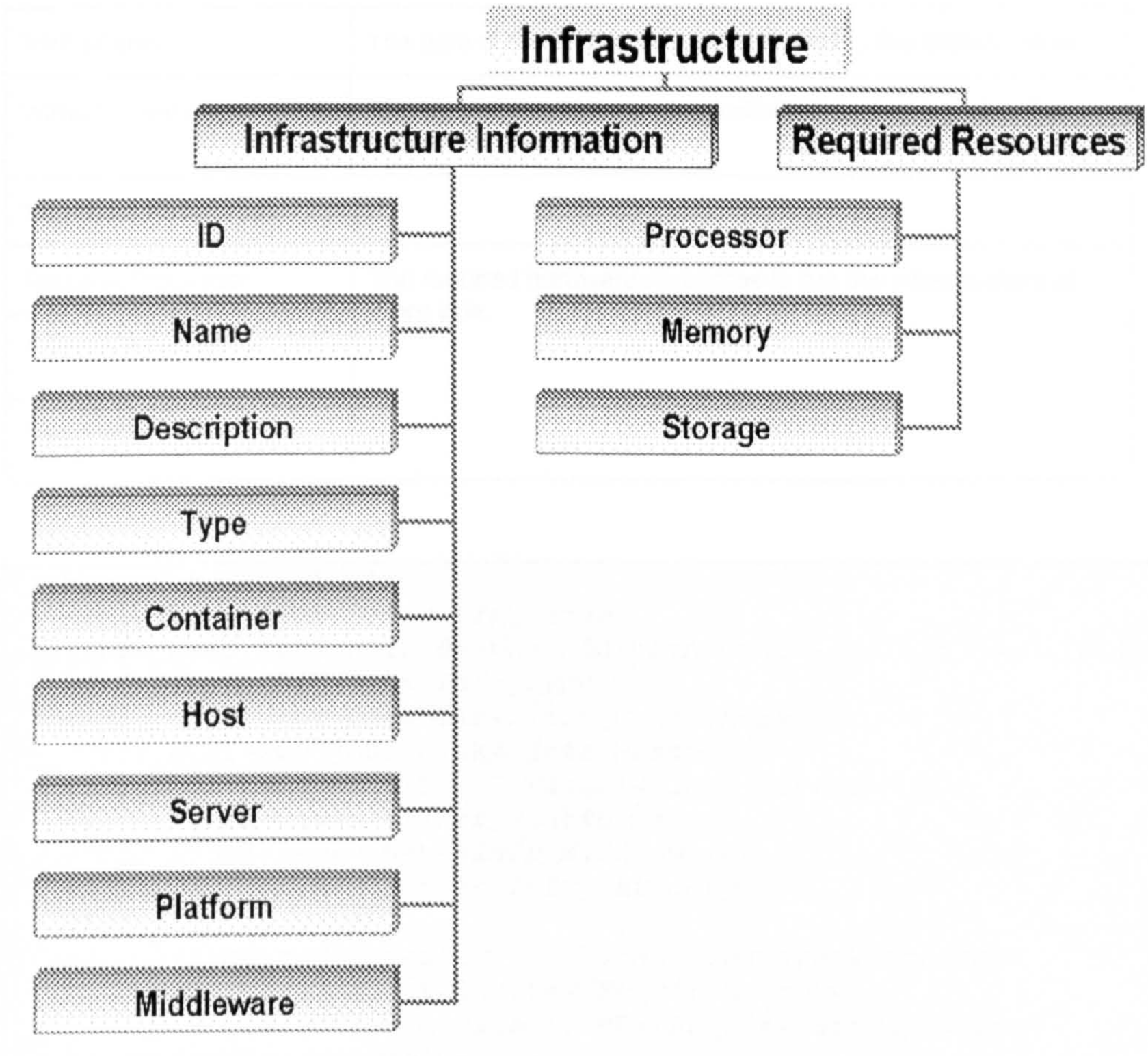


Figure 8.5: Infrastructure's Tags

Table 8.3: Infrastructure's Tags

Elements Name	Description
Infrastructure Information	
InfrID	The infrastructure's ID which should be unique.
InfrName	The infrastructure name.
InfrDescription	The description of the infrastructure.
InfrType	The type of the infrastructure. The type is represented as hardware, software, communication or networking.
InfrContainer	The container that includes the described infrastructure.
InfrHost	The host that is utilised to deploy the described infrastructure. The infrastructure can be deployed in more than one host.
InfrServer	The type of server that is used to deploy such infrastructure. The server can be Unix, windows, Novell, etc....

InfrPlatform	The type of the platform required to deploy the infrastructure.
InfrMiddleware	The type of middleware responsibilities for looking after the designated infrastructure.
Required Resources	
RequiredProcessor	The required hardware resources to run the infrastructure at client side.
RequiredMemory	
RequiredStorage	

```
<Infrastructure Inf-ID="1">
        <Infr_Name>Addition</Infr_Name>
        <Infr_Description>To do the addition process</Infr_Description>
        <Infr_Type>Software</Infr_Type>
        <Infr_Container>cmswomar</Infr_Container>
        <Infr_Host>www.jmu.ac.uk</Infr_Host>
        <Infr_Server>Apache/1.3.0 (Unix)</Infr_Server>
        <Infr_Platform>any</Infr_Platform>
        <Infr_Middleware>.Net</Infr_Middleware>
        <Infr_Catogery>Addition</Infr_Catogery>
        <RequiredResources>
                <RequiredProcessor>PI 233MHz</RequiredProcessor>
                <RequiredMemory>8Mbyte</RequiredMemory>
                <RequiredStorageSize>1.4MB</RequiredStorageSize>
        </RequiredResources>
        <Infra_Contract>
                <ContractID>1</ContractID>
                <ServicesContractName>CMPWOMAR</ServicesContractName>
                <ServicesContractLease>1/1/2007</ServicesContractLease>
        </Infra_Contract>
</Infrastructure>
```

Figure 8.6: Infrastructure Example

8.3. Sensor and Actuator Description Language

As described in the previous chapter, a semantic way for interoperable information between the monitor resources providers and the framework is required to make widely used for such framework in monitoring and control field. Therefore, Sensor and Actuator Description Language (SADL) is developed and implemented to interchange information inside the framework and outside with the resource providers. Different information is required to be published within the framework for describing the utilisation, behaviour and interfaces of the resources. The information for the sensors and actuator is categorized into six major categories namely; sensor

information, actuator information, contract, interface, required resources and application as shown in Figure 8.7. The details of the sensor published information are shown in Table (8.4).

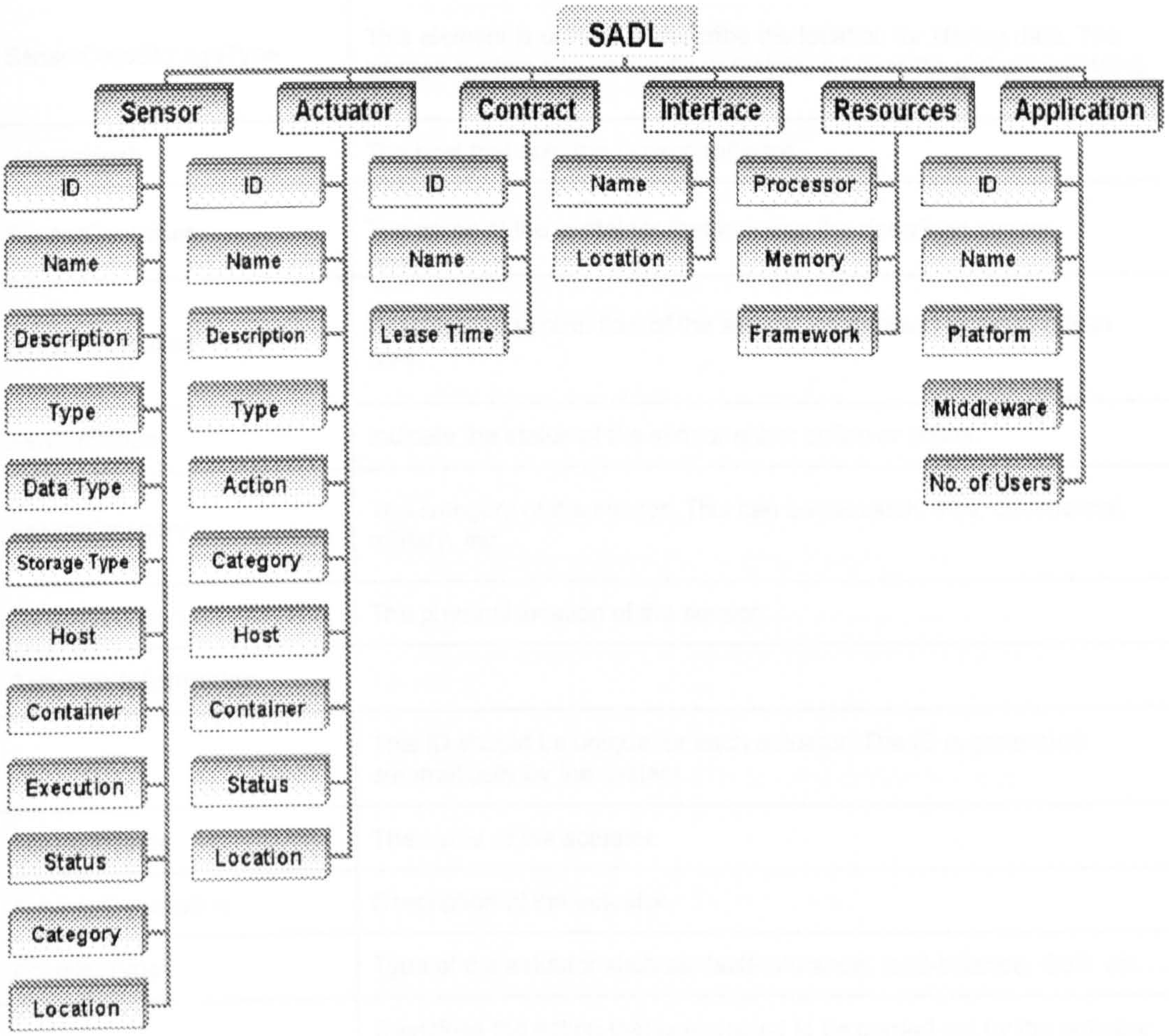


Figure 8.7: SADL Model

Table 8.4: SADL Tags

Elements Name	Comments
Sensor Information	
SensorID	This ID should be unique for each sensor. The ID is generated automatically by the system.
SensorName	The name of the sensor.
SensorDescription	Description of the sensor.

SensorType	Type of the sensor such as; performance, security, Load balance, etc....
SensorDataType	Type of collected data such as; string, integer, object, etc....
SensorDataStorageType	This element is utilised to describe the location for storing data. The storage system can be locally or distributed over the global computing.
SensorHost	The host that hold the sensor software.
SensorContainer	The name of the container that includes the described sensor.
SensorExecution	Describe the control flow of the sensor, on demand or event driven type.
SensorStatus	Indicate the status of the sensor either online or offline.
SensorCategory	The category of the sensor. This can be research, free, commercial, military, etc...
SensorLocation	The physical location of the sensor.
Actuator Information	
ActuatorID	This ID should be unique for each actuator. The ID is generated automatically by the system.
ActuatorName	The name of the actuator.
ActuatorDescription	Description of the actuator.
ActuatorType	Type of the actuator such as; fault-tolerance, load-balance, QoS, etc....
ActuatorAction	Describes the action that is expected to be carried out by the actuator to perform the demanded task. Such actions can be presented as replication service, re-configure system behaviour, re-manage resources, analysis, etc....
ActuatorHost	The host that hold the actuator software or hardware.
ActuatorContainer	The name of the container that includes the described actuator.
ActuatorCategory	The category of the Actuator. This can be research, free, commercial, military, etc...
ActuatorStatus	Indicates the status of the actuator either online or offline.
ActuatorLocation	The physical location of the actuator.
Contract Information	
ContractID	Evaluates the contract between the provider and the framework to

ContractName	check the provider privilege in deploying such resources.
LeaseTime	
Interface Information	
InterfaceName	To describe the available interfaces which can be used to request the service and get the result.
InterfaceLocation	
Resources Information	
ResourcesProcessor	The minimum required resources which are needed to be available for running the monitoring resources.
ResourcesMemory	
ResourcesFramework	
Application Information	
ApplicationID	Application ID
ApplicationName	The name of the application, i.e. Grid, PlanetLab, E-Health, general, etc....
ApplicationMiddleware	The required middleware to be available for running the sensors or actuators.
NoOfUsers	This describes the maximum number of users that can use the resources at the same time. This tag is proposed to ensure a high performance services by limiting the number of users to the specify limit.
Platform	The required platform to run the resources.
CurrentUsers	This describes the current users that use the sensor at the requested time. This may be changed dynamically by the system according to the current users.

SADL is utilised to deploy the planetary-scale sensors overlay with the developed SAF. As a paradigm of that, variety types of grid computing sensors are deployed, discovered and invoked with this framework. In addition, SADL is employed to deploy the PlanetLab overlay sensor with our sensor framework, and hence gathering readings from different slices of the PlanetLab. Different types of sensors specialist in PlanetLab are available to be utilized with such framework to perform the demanded task such as; CoMon [120, 121], Ganglia [39], Node List [124] and others.

Figure 8.8 presents an example of using SADL to deploy a memory sensor for gathering information from different types of targets. The deployed information consists of: basic sensor information, contract information, which describes Service

Level Agreements (SLA), a user interface for requesting sensors (if possible), environment information, and required resource. Figure 8.9 demonstrates an example of deploying an actuator with SADL. Actuator information is deployed with SADL in order to provide an information layer for providing the consumers with the required knowledge in selecting the resources.

```
<SADL>
    <Sensor>
        <SensorID>10</SensorID>
        <SensorName>JMUSensor</SensorName>
        <SensorDescription>To gather processor, memory, and
storage readings</SensorDescription>
        <SesnsorInformation>
            <Type>Performance</Type>
            <DataType>string</DataType>
            <DataStorageType>XML</DataStorageType>
            <Host>http://cms.livjm.ac.uk</Host>
            <Container>CMPwomar</Container>
            <Execution>OnDemand</Execution>
            <Status>On-Line</Status>
            <Category>Memory</Category>
            <Location cmpwomar/sensor</Location>
        </SesnsorInformation>
        <Contract>
            <ContractID>16</ContractID>
            <CntractName>Wail</CntractName>
            <ContractLease>1/1/2006</ContractLease>
        </Contract>
        <Interface>
            <InterfaceName>.Net Sensors</InterfaceName>
            <InterfaceLocation>www.cmpwomar.livjm.ac.uk/
Sensor.exe</InterfaceLocation>
        </Interface>
        <Resources>
            <Processor>PI233</Processor>
            <Memory>32M</Memory>
            <Framework>Windows</Framework>
        </Resources>
        <Application>
            <ApplicationID>1</ ApplicationID>
            <ApplicationName>General</ApplicationName>
            <ApplicationMiddleware>.Net</ApplicationMiddleware>
            <Platform>Any</Platform>
            <Middleware>.Net</Middleware>
            <NoOfUsers>50</NoOfUsers>
            <CurrentUsers>10</CurrentUsers >
        </Application>
    </Sensor>
</SADL>
```

Figure 8.8: SADL Example for Deploying Memory Sensor

```
<SADL>
      <Actuator>
            <ActuatorID>1</ActuatorID>
            <ActuatorName>JMU Actuator</ActuatorName>
            <ActuatorDescription>To overcome the problem of fault
tolerance   according to overload. Generating replication service is
a way that is used in this    actuator</ActuatorDescription>
            <Type>Fault Tolerance</Type>
            <Action>Genrate Replication Service</Action>
            <Host>http:\\www.cms.livjm.ac.uk\cmpwomar\software</Host>
            <Container>cpmwomar</Container>
            <Status>Online</Status>
            <Category>Actuator Software</Category>
            <Location>planetlab1.cambridge.intel-research.net
            </Location>
      </Actuator>
</SADL>
```

Figure 8.9: SADL Example for Deploying Actuator

8.4. Monitor Session Description Language

Yet another, interaction language is required between the consumers, Serviceware and its components (autonomic computing service) and/or control system in one side and the SAF in the other side, as described in Chapter 7. This proposed description language describes the complete information regarding the required monitoring resources, applications and monitored targets. This language is known as Monitor Session Description Language (MSDL). The required information is generated by the requesters in order to be sent to the framework for performing target diagnosis and/or action. The requesters can demand more than one resource at the time to be injected in one or more targets. The framework analyses the requester's demands to find the most appropriate resources. The MSDL is categorized into five parts as shown in Figure 8.10, these parts are:

- **Application section:** concerns with the type of application. This is important to assist the framework to prepare resources that are required for each application like e-health, connected-home machine, Telematics or other applications.
- **Resources section:** concern with the type of resources demanded by the consumer.
- **Target section:** describes the targets which are required for diagnostic or action execution processes.

- **Duration section:** describes the time of diagnoses or action.
- **Contract section:** to exams the privilege of the consumer in utilising such resources.

The details of MSDL are illustrated in Table 8.5.

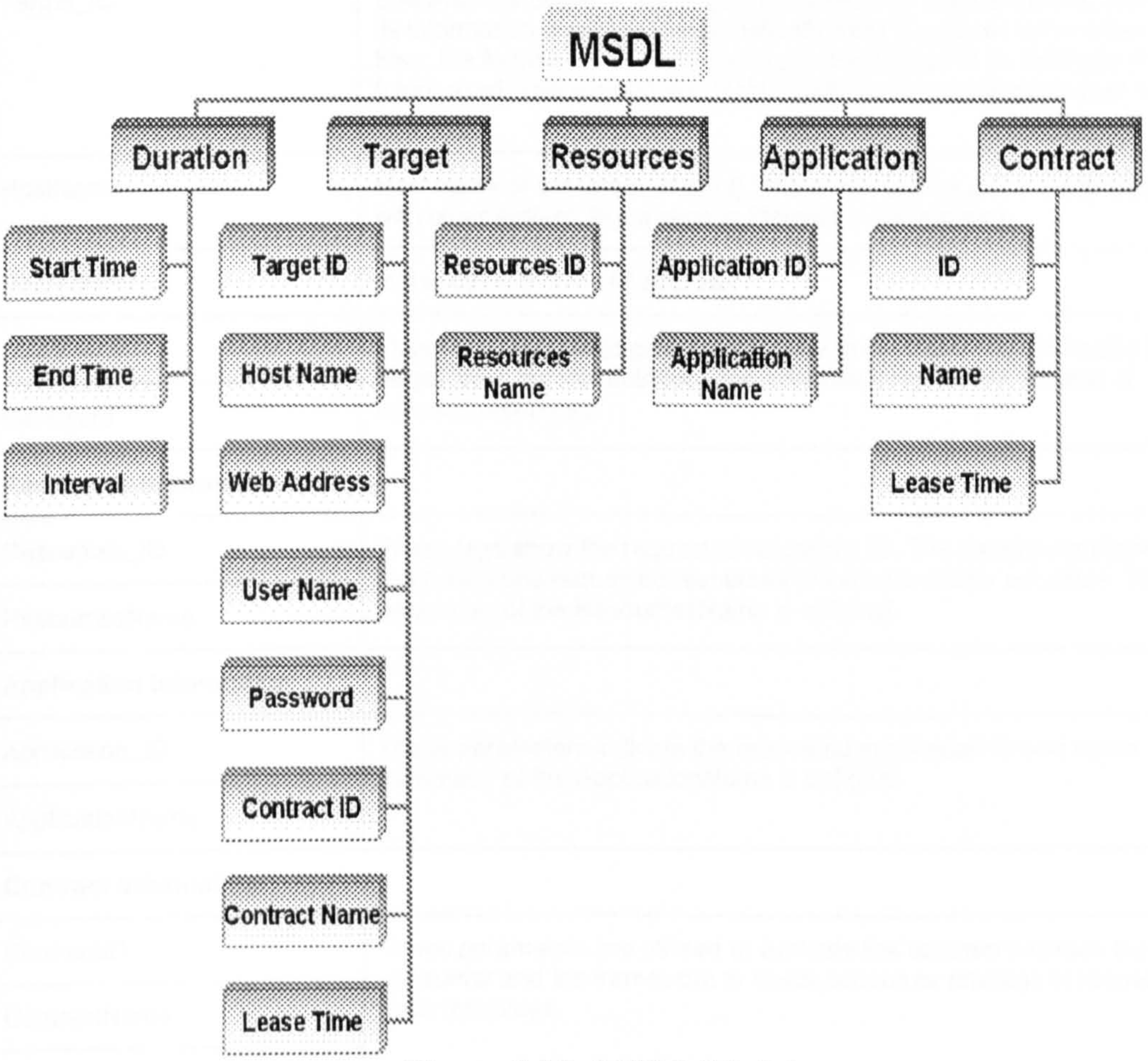


Figure 8.10: MSDL Model

Table 8.5: MSDL Parameters.

Elements Name	Comments
MSDL_ID	This ID should be unique for each requested session. The ID is generated automatically by the system.
Duration Information	
StartTime	Specifies the beginning time for injecting resources in the targets.
EndTime	Specifies the time for destroying the injected resources inside the client

Duration	This parameter describes the interval between process of gathering reading or carrying out an action and the other process. This parameter is helpful in reducing the amount of data transfer through the network by specifying the time required for each process.
Targets Information	
Target_ID	Specifies the target. If the target registered with the framework, then all its information is retrieved automatically from the stored information. Else, the framework will add the target information to its database for future used. The session can include information about more than one target.
HostName	Host name of the target (if exist). The target can be a unit integrated with other system, like a slice in PlanetLab environment.
WebAddress	In this case, it is the IP address.
UserName	These are required to give the authority to the framework to invoke the target. With out this authority, the framework rejects the request of injecting resources.
Password	
Resources Information	
Resources_ID	These tags show the requested resources ID. The session can include more than one requested resources of sensors and/or actuators. The parameter of the ResourcesName is optional.
ResourcesName	
Application Information	
Application_ID	These parameters indicate the requested application ID and name. The parameter of the ApplicationName is optional.
ApplicationName	
Contract Information	
ContractID	These parameters are utilised to evaluate the contract between the consumer and the framework to check consumer privilege in requesting such resources.
ContractName	
LeaseTime	

An example of using MSDL with sensor and actuator framework is shown in Figure 8.11. This example demonstrates the utilising of such description language for requesting memory sensor from sensor and actuator framework. In this example, two sensors are demanded to be injected in one slice of the PlanetLab with duration of two minutes and an interval of 20mSec.

```
<MSDL>
      <MonitorSession Session_ID="10">
            <Application>General</Application>
            <Duration>
                  <DurationStart>11:05:49 AM</DurationStart>
                  <DurationEnd>11:07:49 AM</DurationEnd>
                  <Interval>20000</Interval>
            </Duration>
            <TargetInformaion>
                  <Client_ID>121</Client_ID>
                  <HostName>http://livjm.ac.uk/taleb</HostName>
                  <WebAddress>128.112.139.71</WebAddress>
                  <UserName>Wail</UserName>
                  <Password>XXXXXX</Password>
            </TargetInformaion>
            <ContractInformation>
                  <ContractID>16</ContractID>
                  <ContractName>Wail</ContractName>
                  <LeaseTime>1/1/2006</LeaseTime>
            </ContractInformation>
            <Sensors>
                  <ID>10</ID>
                  <Name>JMUSensor</Name>
            </Sensors>
            <Sensors>
                  <ID>12</ID>
                  <Name>MemSensor</Name>
            </Sensors>
      </MonitorSession>
</MSDL>
```

Figure 8.11: MSDL Example for Requesting Memory Sensor

8.5. Summary

This chapter introduces three types of description languages in order to provide a semantic utility to support the autonomic middleware interaction model. The first one is Assembly Services and Infrastructures Description Language (ASIDL) developed to provide an open standard metamodel interchange between resources providers and ASIF. In addition, ASIDL is used to interchange information between consumer and middleware complex. The second description language is Sensor and Actuator Description Language (SADL), which is developed in order to manage the deployment, discovery and invocation of monitoring and actuation resources with SAF. The third description language is Monitor Session Description Language

(MSDL), which is generated by the consumer in order to specify/define monitoring request to the sensor and actuator framework.

Overall, this chapter details the developed description languages and their semantic support for the proposed self-management middleware services. This next chapter test these designs through a set of quantitative and qualitative evaluations.

CHAPTER 9

EVALUATION

9.1.Introduction

This chapter presents an evaluation of the proposed self-management middleware services and associated utilities and frameworks. As acknowledged by many [7, 93, 154], such an evaluation is a challenging task as there are no known/clear metrics or accepted benchmarks [154] to evaluate autonomic system.

However, this evaluation adopts both quantitative and qualitative analysis of a set of implemented self-managing software prototypes, which illustrates either; the performance profile of systems with and without the proposed self-managing features, or demonstrating the generality of the proposed reference model, associated services, and frameworks.

The remainder of the chapter outlines the evaluation methodology, followed by quantitative and qualitative evaluations of the work. Finally, the chapter concludes with a critical analysis and general discussion of the results.

9.2.Methodology

The evaluation has been designed to demonstrate the use and effect of the developed and implemented of the autonomic computing capabilities for the self-management model from both quantitative and qualitative perspectives. In other words, we will analyse the effect of the autonomic computing behaviour on the prototype software performance overhead, which is, for convenience restricted to fidelity and processing time.

9.2.1. Objectives

The main objective of the evaluation process is to show the feasibility of using the developed models and technologies with the distributed enterprise applications. To achieve this goal, three test example applications have been developed utilising autonomic computing services, developed frameworks and description languages. These test applications are: on-demand services, remote e-health monitoring system and monitoring and controlling PlanetLab environment. These applications represent the evaluation from a qualitative perspective. As for the quantitative evaluation, an experiment is set up to measure and compare the response time of the same testbed application with and without autonomic capabilities. Comparison between the sets of readings is merely used to exhibit a comparative performance profiling, not full system performance evaluation study which is out of the scope of this research.

9.2.2. Approach

Although this evaluation is not intended to be a formal performance evaluation of our self-management model and its associated software, nevertheless, we use elapsed time as a performance profile metric to outline the effect and overheads of *ad-hoc* autonomic computing capabilities on systems' performance.

For calibration purposes, prior to the evaluation, a range of preliminary experiments have been conducted including:

- Designing, developing and implementing a number of dummy services for the purpose of measuring the response time.
- Designing, developing and implementing software for generating training data for the intelligent services training and test phase.
- Measuring the access time to the service in LAN and WAN environment.

9.2.3. Environment

The evaluation of this system has been performed using Dell 510m, Centrino (tm) ~ 1.7 GHz processor with 512 MB of memory, and running MS Windows XP Professional operating system with service pack 2 and connected via Ethernet. The

example applications were implemented using VS.Net programming language with .Net framework 1.1. .

9.3.The Quantitative Evaluation

The autonomic computing service is employed in the experiment is the advance reservation and assembly of a set of required services – a kind of reservation of services. For the experiment a number of services are generated in order to determine the required response time to invoke these services by the consumers. Such services can represent calculator, dictionary, accounting service, e-government services and other types of services. The response time consists of transmission, propagation, queuing and processing times. The processing time is much higher than the other type of times according to the use of fast connection with fast transmission devices. Therefore, in our case we are considering the processing time as a response time.

9.3.1.Requesting Service without Autonomic Computing Capability

The first phase of the experimental is to request services without utilising the autonomic computing service. In this stage, the consumer accesses to the required services directly without preparing in advanced for the aimed services by the middleware system. In this experimental, we make the consumer requests 20 services from the resources container. These requests are stored in XML format, as shown in Figure 9.1, in order to be used later in training the autonomic computing service for predicting the required services for that consumer. The developed monitoring system is employed to track the access to the required services and determine the processing time for accessing them, as shown in Table 9.1. The total response time for requesting 20 services, which is equal to the summation of the twenty's response time, is (209 seconds). In addition, Table 9.1 demonstrates further information that assists later in training the reservation of the services in advance. This information is: User ID, sequence of the requested service, service ID and name, Start and end time and response time.

```
<?xml version="1.0" standalone="yes" ?>
<Readings>
      <ReadingSet>
            <UserID>6</UserID>
            <Sequence>0</Sequence>
            <ServiceID>5</ServiceID>
            <ServiceName>S5</ServiceName>
            <StartTime>10/24/2005 8:15:53 AM</StartTime>
            <EndTime>10/24/2005 8:16:05 AM</EndTime>
            <Duration-Sec>12.2</Duration-Sec>
      </ReadingSet>
      <ReadingSet>
            <UserID>6</UserID>
            <Sequence>1</Sequence>
            <ServiceID>12</ServiceID>
            <ServiceName>S12</ServiceName>
            <StartTime>10/24/2005 8:16:09 AM</StartTime>
            <EndTime>10/24/2005 8:16:19 AM</EndTime>
            <Duration-Sec>10</Duration-Sec>
      </ReadingSet>
      <ReadingSet>
            <UserID>6</UserID>
            <Sequence>2</Sequence>
            <ServiceID>9</ServiceID>
            <ServiceName>S9</ServiceName>
            <StartTime>10/24/2005 8:16:24 AM</StartTime>
            <EndTime>10/24/2005 8:16:36 AM</EndTime>
            <Duration-Sec>13</Duration-Sec>
      </ReadingSet>
</Readings>
```

Figure 9.1: Services Requests by the Consumer

Table 9.1: Response Time With Out Using Autonomic Computing Service

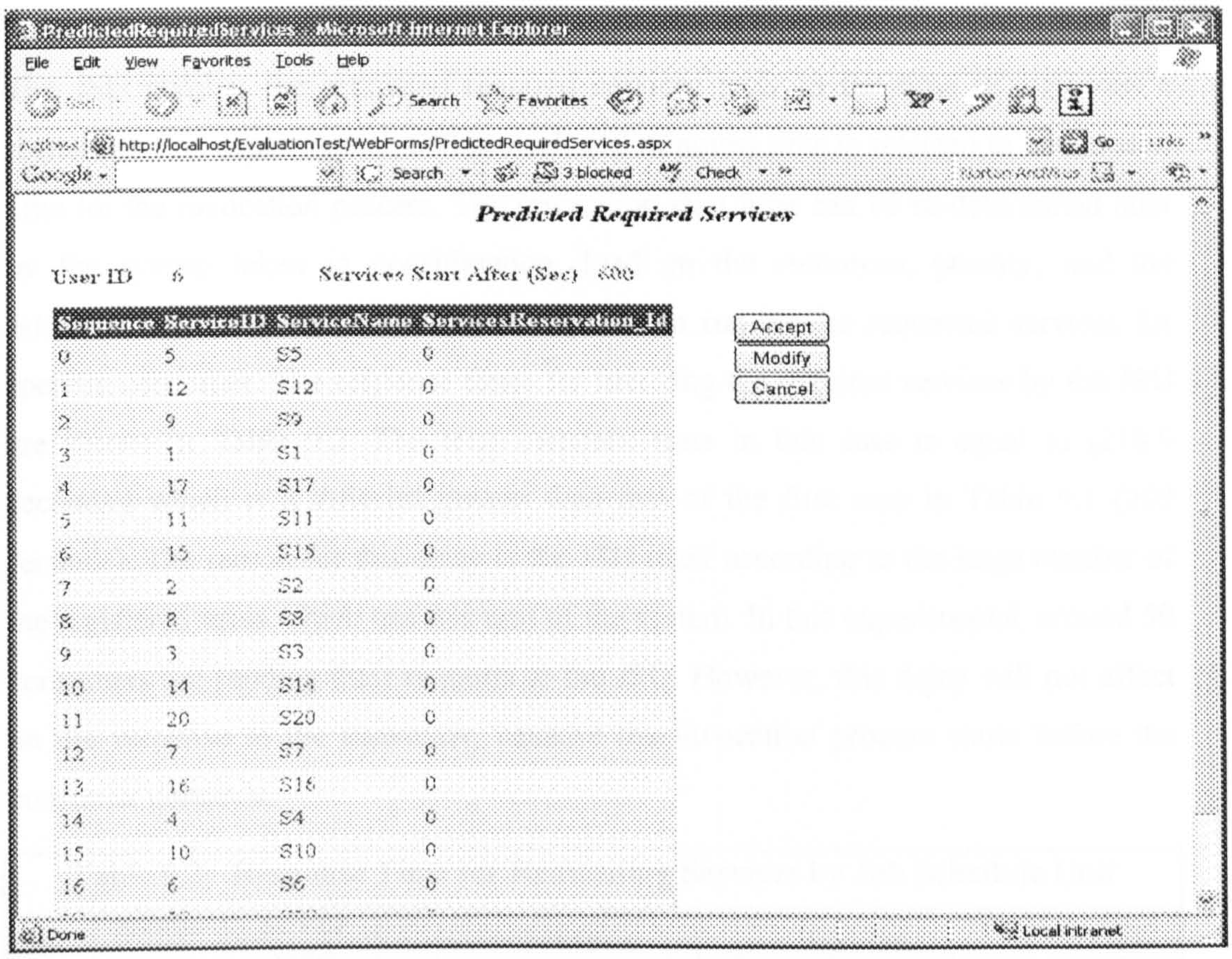
User ID	Seq.	Service ID	Service Name	Start Time	End Time	Response Time (Sec)
6	0	5	S5	10/24/2005 8:15:53 AM	10/24/2005 8:16:05 AM	12.2
6	1	12	S12	10/24/2005 8:16:09 AM	10/24/2005 8:16:19 AM	10
6	2	9	S9	10/24/2005 8:16:24 AM	10/24/2005 8:16:36 AM	13
6	3	1	S1	10/24/2005 8:16:43 AM	10/24/2005 8:16:53 AM	11.4
6	4	17	S17	10/24/2005 8:16:58 AM	10/24/2005 8:17:08 AM	10
6	5	11	S11	10/24/2005 8:17:19 AM	10/24/2005 8:17:26 AM	7.2
6	6	15	S15	10/24/2005 8:17:46 AM	10/24/2005 8:17:53 AM	6
6	7	2	S2	10/24/2005 8:17:56 AM	10/24/2005 8:18:06 AM	10
6	8	8	S8	10/24/2005 8:18:11 AM	10/24/2005 8:18:24 AM	13
6	9	3	S3	10/24/2005 8:18:32 AM	10/24/2005 8:18:42 AM	11.6
6	10	14	S14	10/24/2005 8:18:50 AM	10/24/2005 8:18:53 AM	3
6	11	20	S20	10/24/2005 8:18:59 AM	10/24/2005 8:19:06 AM	6
6	12	7	S7	10/24/2005 8:19:23 AM	10/24/2005 8:19:30 AM	7
6	13	16	S16	10/24/2005 8:19:39 AM	10/24/2005 8:19:51 AM	12.5
6	14	4	S4	10/24/2005 8:19:57 AM	10/24/2005 8:20:10 AM	13
6	15	10	S10	10/24/2005 8:20:13 AM	10/24/2005 8:20:23 AM	9
6	16	6	S6	10/24/2005 8:20:42 AM	10/24/2005 8:20:55 AM	13.3

6	17	18	S18	10/24/2005 8:21:08 AM	10/24/2005 8:21:21 AM	14
6	18	13	S13	10/24/2005 8:21:27 AM	10/24/2005 8:21:39 AM	13
6	19	19	S19	10/24/2005 8:21:47 AM	10/24/2005 8:21:59 AM	13.8
					Total Response Time	**209 seconds**

9.3.2. Requesting Service with Autonomic Computing Capability

In the next stage of the experimental, the autonomic computing service is adopted to carry out the job of *Service Reservation Unit (SRU)* and *Job Schedule Unit (JSU)* in order to improve the self-management system by providing an automated mechanism for running the required services prior to the consumers' requests. The developed system represents an intermediate layer between the consumer and the distributed resource. In this way, the response time is expected to be reduced according to the in advanced invocation of the services. The autonomic computing service predicts the required services for each consumer depending on his/her/its requesting history. The process of predicting and creating a profile for each user is known as S*ervice Reservation Unit (SRU)*. Intelligent classification is assumed to be used in order to sort out the required services for each consumer. In the coming sections, machine learning algorithms and mathematic analysis are presented in order to perform the prediction and intelligent classification. For the experimental purpose, the same monitored requested services are considered to be the predicted profile list for the selected consumer.

Figure 9.2, illustrates a basic user interface for the SRU list, showing a predicted/learned list of required software services and their usage sequence for a user ID number 6 (Fig. 9.3). This interface enables the users to accept or modify the system predicted services to be reserved and used on-demand. Hence, it assists the machine learning service to tune and/or reinforce its predicted/learned model.



Sequence	ServiceID	ServiceName	ServicesReservation_Id
0	5	S5	0
1	12	S12	0
2	9	S9	0
3	1	S1	0
4	17	S17	0
5	11	S11	0
6	15	S15	0
7	2	S2	0
8	8	S8	0
9	3	S3	0
10	14	S14	0
11	20	S20	0
12	7	S7	0
13	16	S16	0
14	4	S4	0
15	10	S10	0
16	6	S6	0

Figure 9.2: User Interface for Consumer Predict Service Profile

```
<?xml version="1.0" encoding="utf-8" ?>
<PredictedRequiredServices>
      <ServicesReservation>
            <UserID>6</UserID>
            <StartAfter-Sec>600</StartAfter-Sec>
            <ServiceSet>
                  <Sequence>0</Sequence>
                  <ServiceID>5</ServiceID>
                  <ServiceName>S5</ServiceName>
            </ServiceSet>
            <ServiceSet>
                  <Sequence>1</Sequence>
                  <ServiceID>12</ServiceID>
                  <ServiceName>S12</ServiceName>
            </ServiceSet>
            <ServiceSet>
                  <Sequence>2</Sequence>
                  <ServiceID>9</ServiceID>
                  <ServiceName>S9</ServiceName>
            </ServiceSet>
      </ServicesReservation>
</PredictedRequiredServices>
```

Figure 9.3: Predicted Services List for Consumer ID 6

After that, the list of the predicted services is forward from the SRU unit to the JSU. The JSU invokes the estimated required services prior to the time of consumer's request. In this experimental, 600 seconds (10 minutes) time is selected to be the start time for the invocation process. The invocation start time can be re-determined later by the system taken in consideration, load on the resources, priority, and the estimated total response time that is required to run all the requested services for specific consumer. The response times for invoking the required services by the JSU are shown in Table 9.2. The total response time in this case is equal to (218.9 seconds.) which is a little bit greater than that of the first case in Table 9.1 (209 seconds). The reason for this delay is the JSU itself according to the large number of the registered users which use this unit of the system. In this experimental, around 50 consumers are sending their requests to the JSU. However, this delay will not affect on the response to the consumer, because the invocation process starts before the consumer demands.

Table 9.2: Response Time for Requesting Services by Job Schedule Unit

Seq.	Service ID	Service Name	Start Time	End Time	Duration (Sec)
0	5	S5	10/24/2005 11:11:35 AM	10/24/2005 11:11:48 AM	12.6
1	12	S12	10/24/2005 11:11:48 AM	10/24/2005 11:11:59 AM	10.8
2	9	S9	10/24/2005 11:11:59 AM	10/24/2005 11:12:11 AM	12.4
3	1	S1	10/24/2005 11:12:11 AM	10/24/2005 11:12:22 AM	10
4	17	S17	10/24/2005 11:12:22 AM	10/24/2005 11:12:33 AM	11.8
5	11	S11	10/24/2005 11:12:34 AM	10/24/2005 11:12:39 AM	5
6	15	S15	10/24/2005 11:12:39 AM	10/24/2005 11:12:46 AM	7.8
7	2	S2	10/24/2005 11:12:46 AM	10/24/2005 11:12:58 AM	12
8	8	S8	10/24/2005 11:12:58 AM	10/24/2005 11:13:13 AM	14.5
9	3	S3	10/24/2005 11:13:13 AM	10/24/2005 11:13:22 AM	9
10	14	S14	10/24/2005 11:13:22 AM	10/24/2005 11:13:25 AM	3
11	20	S20	10/24/2005 11:13:25 AM	10/24/2005 11:13:34 AM	8.3
12	7	S7	10/24/2005 11:13:34 AM	10/24/2005 11:13:43 AM	9
13	16	S16	10/24/2005 11:13:43 AM	10/24/2005 11:13:55 AM	11.8
14	4	S4	10/24/2005 11:13:55 AM	10/24/2005 11:14:10 AM	15.8
15	10	S10	10/24/2005 11:14:10 AM	10/24/2005 11:14:22 AM	11
16	6	S6	10/24/2005 11:14:22 AM	10/24/2005 11:14:33 AM	11
17	18	S18	10/24/2005 11:14:33 AM	10/24/2005 11:14:50 AM	16.7
18	13	S13	10/24/2005 11:14:50 AM	10/24/2005 11:15:03 AM	12.8
19	19	S19	10/24/2005 11:15:03 AM	10/24/2005 11:15:16 AM	13.6
				Total Response Time	**218.9 seconds**

When the consumer starts requesting the services, most of the results are available at the user profile in the JSU. To be in the safe side regarding the predicted list, we assumed that 5 of the requested services by the consumer are out of the sequence of those which are expected by the SRU in the previous stage. Therefore, the JSU needs to forward the consumer requests which are not match to the predicted profile list to the original services and get the result and then return back the result to the consumer. The response times for this process are shown in Table 9.3 and part of the XML file for JSU and requesting services by the consumer is shown in Figure 9.4.

Table 9.3: Response Time With Using Autonomic Computing Service

Seq.	Service ID	Service Name	Start Time	End Time	Duration (Sec)
0	5	S5	10/24/2005 11:22:35 AM	10/24/2005 11:22:38 AM	3.2
1	12	S12	10/24/2005 11:22:38 AM	10/24/2005 11:22:40 AM	2.2
2	9	S9	10/24/2005 11:22:40 AM	10/24/2005 11:22:42 AM	1.7
3	1	S1	10/24/2005 11:22:42 AM	10/24/2005 11:22:45 AM	3
4	17	S17	10/24/2005 11:22:45 AM	10/24/2005 11:22:46 AM	1
5	11	S11	10/24/2005 11:22:46 AM	10/24/2005 11:22:48 AM	1.7
6	3	S3	10/24/2005 11:22:48 AM	10/24/2005 11:22:58 AM	9.7
7	2	S2	10/24/2005 11:22:58 AM	10/24/2005 11:23:00 AM	2
8	8	S8	10/24/2005 11:23:00 AM	10/24/2005 11:23:02 AM	2
9	20	S20	10/24/2005 11:23:02 AM	10/24/2005 11:23:12 AM	10
10	14	S14	10/24/2005 11:23:12 AM	10/24/2005 11:23:13 AM	1
11	15	S15	10/24/2005 11:23:13 AM	10/24/2005 11:23:20 AM	8.2
12	7	S7	10/24/2005 11:23:20 AM	10/24/2005 11:23:22 AM	2
13	16	S16	10/24/2005 11:23:22 AM	10/24/2005 11:23:25 AM	3
14	10	S10	10/24/2005 11:23:25 AM	10/24/2005 11:23:37 AM	12
15	4	S4	10/24/2005 11:23:37 AM	10/24/2005 11:23:53 AM	16
16	6	S6	10/24/2005 11:23:53 AM	10/24/2005 11:23:55 AM	2
17	18	S18	10/24/2005 11:23:55 AM	10/24/2005 11:23:57 AM	2
18	13	S13	10/24/2005 11:23:57 AM	10/24/2005 11:23:58 AM	1
19	19	S19	10/24/2005 11:23:58 AM	10/24/2005 11:24:01 AM	3
				Total Response Time	**86.7 seconds**

```
<?xml version="1.0" standalone="yes" ?>
<SystemWithAC>
      <ServicesRequest>
            <UserID>6</UserID>
            <JobSchedule>
                  <Sequance>0</Sequance>
                  <ServiceID>5</ServiceID>
                  <ServiceName>S5</ServiceName>
                  <StartTime>10/24/2005 11:11:35 AM</StartTime>
                  <EndTime>10/24/2005 11:11:48 AM</EndTime>
                  <Interval>12.6</Interval>
            </JobSchedule>
            <UserRequest>
                  <Sequance>0</Sequance>
                  <ServiceID>5</ServiceID>
                  <ServiceName>S5</ServiceName>
                  <StartTime>10/24/2005 11:22:35 AM</StartTime>
                  <EndTime>10/24/2005 11:22:38 AM</EndTime>
                  <Interval>3.2</Interval>
            </UserRequest>
            <JobSchedule>
                  <Sequance>1</Sequance>
                  <ServiceID>12</ServiceID>
                  <ServiceName>S12</ServiceName>
                  <StartTime>10/24/2005 11:11:48 AM</StartTime>
                  <EndTime>10/24/2005 11:11:59 AM</EndTime>
                  <Interval>10.8</Interval>
            </JobSchedule>
            <UserRequest>
                  <Sequance>1</Sequance>
                  <ServiceID>12</ServiceID>
                  <ServiceName>S12</ServiceName>
                  <StartTime>10/24/2005 11:22:38 AM</StartTime>
                  <EndTime>10/24/2005 11:22:40 AM</EndTime>
                  <Interval>2.2</Interval>
            </UserRequest>
      </ServicesRequest>
</SystemWithAC>
```

Figure 9.4: Predicted Services List for Consumer ID 6

From table 9.3, the total response time for requesting 20 services is equal to (86.7 seconds), which means the improvement ratio is equal to:

$$improvmentRatio = \frac{(Total\ \mathrm{Re}sponseWithoutAC - Total\ \mathrm{Re}sponseTimeWithAC)}{Total\ \mathrm{Re}sponseWithoutAC} * 100$$

$$improvmentRatio = \frac{(209 - 86.7)}{209} * 100 = 58.5\%$$

This improvement ratio (58.5%) is a result of utilising SRU and JSU based on autonomic computing unit with 5 mismatch hits for the consumer requests. Figure 9.5

illustrates the differences in response time between the two systems (with and without autonomic computing). The dashed line represents the system with autonomic computing capabilities, while straight line represents the system without autonomic computing. From this figure, 5 peak values are shown for the system with autonomic computing, which represents the time that the system needs to invoke the original service instead of utilising the ready results.

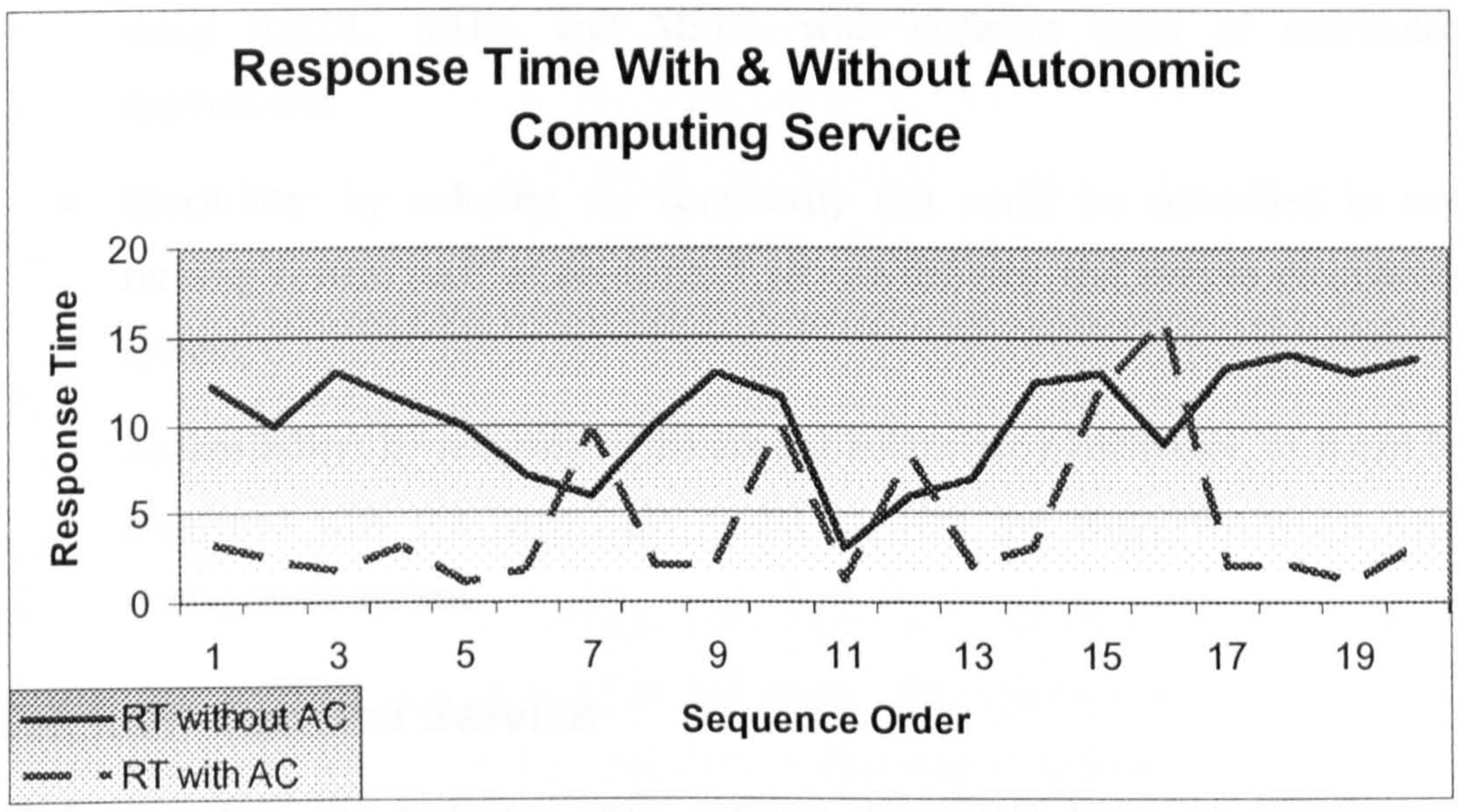


Figure 9.5: Comparison of the Two Systems (With and Without Autonomic Computing Service)

9.4. The Qualitative Evaluation

This section describes a general evaluation of the self-management system, which is relying on embedding autonomic computing capabilities and their frameworks and tools into enterprise distributed applications. The qualitative evaluation is expressed on the basis of employing three applications and scenarios. These applications and scenarios use the autonomic computing in achieving self-management system. On-Demand Services, remote e-health monitoring system and monitoring and controlling PlanetLab environment are the three applications which are adopted in this evaluation. A number of qualitative metrics are proposed in these scenarios, which are:

- **Functionality:** by enabling the distributed application service to be delivered on-demand, monitored and controlled within self-management middleware system.
- **Generality:** by designing a general structural capability and model that is not bounded or limited to any specific system or case study and not tied to any specific programming language or technology. And this has been done by using ASIDL, SADL and MSDL with different types of distributed applications.
- **Flexibility**: by reducing the complexity that could be embedded in any running system such as on-demand service delivery and remote monitoring system.
- **Extensibility:** by referring to the control autonomic middleware, as it can be combined with the large-scale model to fulfil the requirements for self-management system.

9.4.1. On-Demand Service

Grid computing and planetary scale systems necessitate range of management processes for shrinking the interaction between the consumers and services/infrastructures. In addition, management processes leverage the QoS in prospective of response time, fidelity and reliability as mentioned before. On-Demand Service (ODS) is another way to reduce the interaction between the users and the resources by adjusting the resources to give superior services to the users. ODS is build on autonomic computing services to manage the consumers' usages. ODS is considered as one of the middleware services. Taking benefit of the concept of service-oriented model for the grid computing, the consumer should consider his requirements in advance before sending requests to the middleware for invoking demanded services. The process of creating an advanced ODS requests assesses in reducing the unnecessary and redundant invocation processes, and hence affects in reducing response time. A Services Reservation Unit (SRU) is much more agreeable

in this case to manage the process of an advanced ODS. Therefore the middleware needs to be more specific in the provision of the services required.

Self–management system is integrated with ODS to offer an intelligence fabric to the system. The embedded self-management is expressed through variety of autonomic computing capabilities, which are: self-configuration, self-organising, self-protective and self-optimising. These capabilities are offered by serviceware layer as described in Chapter 6. SOM is utilised here to perform an intelligent stuff of classifying consumers for ODS. In which, the SOM service is used to classify the types of consumers according to their respective networked appliances usage model (classes/features) and services dependencies. The models are accessed by our experimental self-managing infrastructure to develop an automated mechanism for deploying and activating the required services. Such processes are occurred in line with learnt/extracted usage models and baseline architecture of specified services federations/assemblies and discovering and activating additional services on-demand. Other types of supervised learning algorithms like regression, decision trees, Bayesian learning, reinforced learning and many others demand more analyzing to achieve prospected goal with minimum error. This would ultimately claims manual procedures and testing until reaching those goals, which in return requires managed centre administration.

ASIDL is adopted here to deploy services and infrastructures with resources container in semantic format. A case study for connected-home machine is presented in this section to show the feasibility of embedding ODS with enterprise applications.

9.4.1.1. On-Demand Service Components

ODS requires an integrations of diversity of components to perform the demanded jobs. These components are shown in Figures 9-6 and 9-7 and summarised in the following points:

- **Consumer:** represents software or human agent requesting a given set of the resources.

- **Request Services Agent:** manages and handles all consumers' requests. A semantic support for metamodel sharing between consumer and the system is used to support open standard interoperation.
- **Discovery Service:** supports services discovery within a given virtual or physical container (host). This provides service for request services container and job schedule service.
- **Monitoring Service:** audits the consumer behaviour, activities and usage. The collected information is provided to the autonomic computing unit for intelligence processes.
- **Autonomic Computing:** offers an intelligent layer to the ODS system. This can provide self-organising, self-configuration, self-protection and self-configuration capabilities.
- **Group:** defines a number of consumers or members which share the same behaviour or patterns.
- **Service Reservation Unit (SRU):** queues and handles all the required services for a given consumers as soon as they are login to the system and prior to services requests. The recorded information includes list of required services, time of operation, and contract information. This unit forwards this information to JSU.
- **Job Schedule Unit (JSU):** schedules the request of services, according to the received information from services reservation unit. The requested services or resources are passed to the discovery unit at the required time of invocation.
- **Resource Container:** contains all deployed resources from services and infrastructures by the providers.

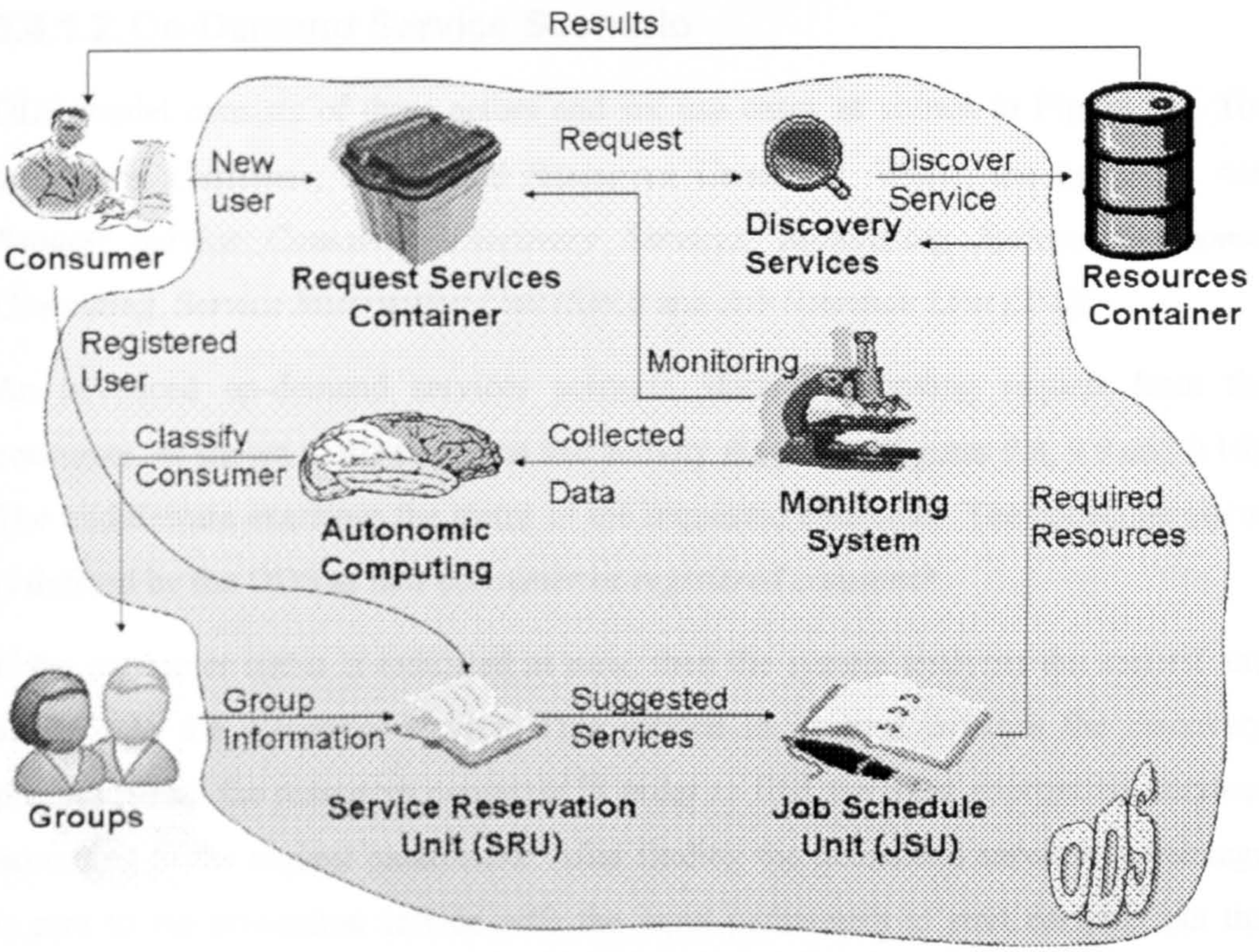


Figure 9.6: On-Demand Service

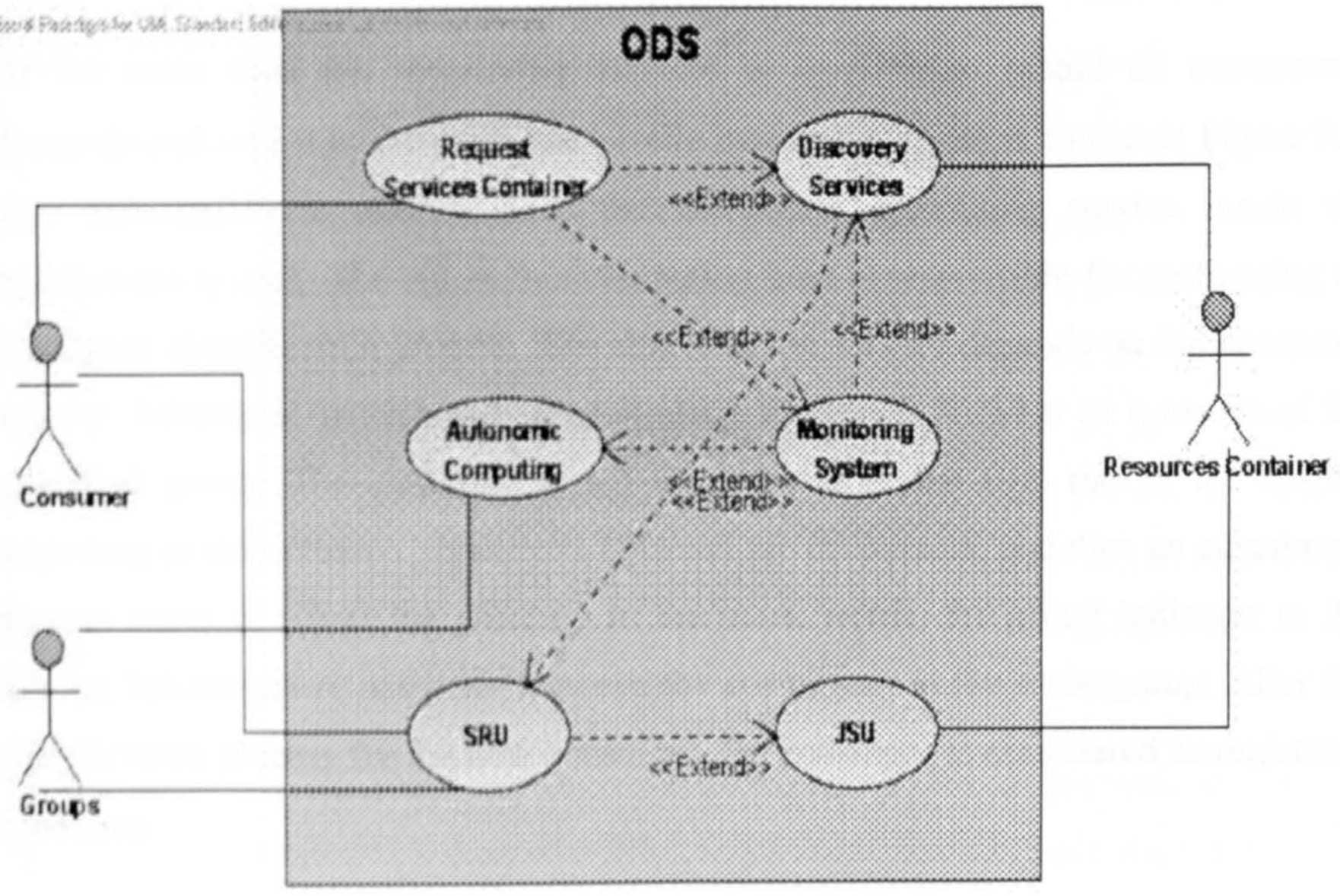


Figure 9.7: ODS UML Use Cases Diagram

9.4.1.2. On-Demand Service Scenario

ODS model consists of three actors and six use cases as shown in Figure 9.7. The actors are: *Customer*, *Group* and *Resources Container*. While, the use cases are: *Request Service Container*, *Discovery Services*, *Monitoring System*, *Autonomic Computing*, *Service Reservation Unit (SRU)* and *Job Schedule Unit (JSU)*.

An advanced on-demand services scenario starts by sending request from the consumer, as shown in the sequence and activity diagrams (Figures 9.8, 9.9 and 9.10). The middleware examines the status of the requested consumer. The consumer status is defined by the ODS as *new consumer* or *registered consumer*.

If the consumer status is indicated as new, then the system analyses the request and sends it to the discovery service. This service in its turn emerges the discovery process inside the resources container in order to find the most appropriate services according to the request parameters. After finding the demanded services, a message is sent to the invocation service with the input parameters to start carrying out the required task. The result of the performing task is presented to the consumer as results or actions.

At the same time the monitoring services is working to record all consumers' demands and save it in the consumer profile inside the logger as shown in Figure 9.8. This information is used to feed the autonomic computing service inside the middleware system. The autonomic computing here is responsible for performing an intelligent classification process. The classification service depends on the consumer activity, behaviour, pattern and services usage to classify him/her or it to one of the classified group. The classified group is clustered at the first run of the system according to the common parameters between the consumers, and then an adjustment process starts to adjust the groups. In the other words, the group indicates to the sharing behaviours or activities between the consumers in the same group. After the classification process for the new consumer, the consumer is considered as registered consumer.

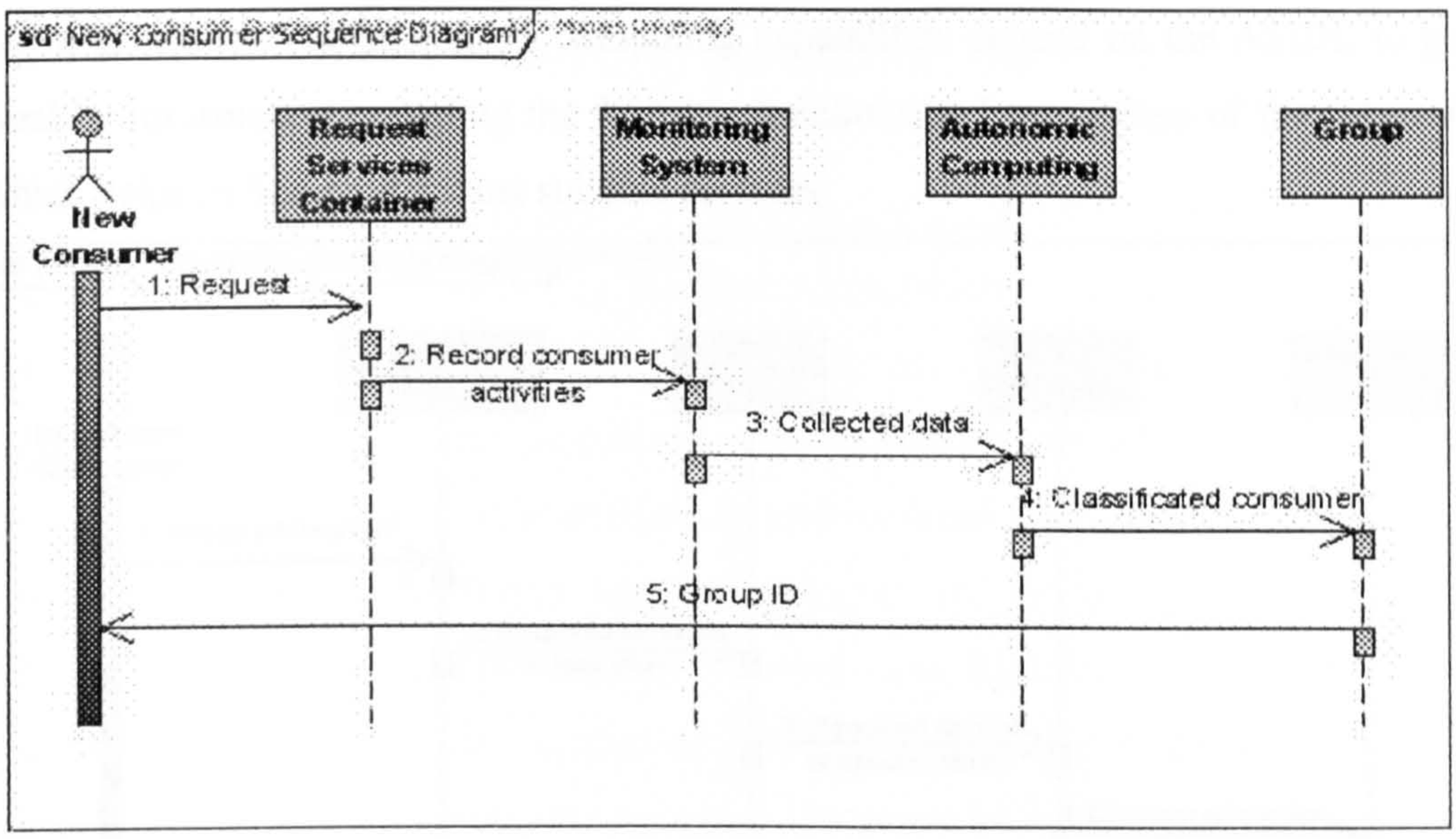


Figure 9.8: ODS-UML Sequence Diagram for Registered Consumer

For the registered consumer, the middleware starts gathering the group information of the requested consumer. The consumer sends his group ID with his demands of the required services to help SRU for arranging the demanded services in advance. The SRU sends a request with all demanded services to the JSU. This request consists of information regarding the time for executing each service, contract, services information, and consumer's information.

The JSU manages the execution of the consumers' demanded tasks. This has been achieved by scheduling the execution of each service. JSU sends the demanded service to the discovery service for starting invocation process. On the other hand, JSU provides the system with in advance information regarding the expected load on each service. This information is utilised to anticipate the load on the services. In this way, the system has the time to establish his plans, strategies and policies to conquer fault tolerance problems according to the overload. Load balance, replication, mirror and other methods can be used in solving such problems of fault tolerance depending on the information that is provided by JSU.

To enhance the operation of discovering services and infrastructures by the middleware in sense of fidelity, QoS, high availability and response time, autonomic computing is proposed to be used to perform the concepts of self-configuration and

self-organising. The autonomic computing capabilities depend on the ASIDL to get wealthy information regarding the features, functions and parameters of the services, which helps in finding the most suitable services.

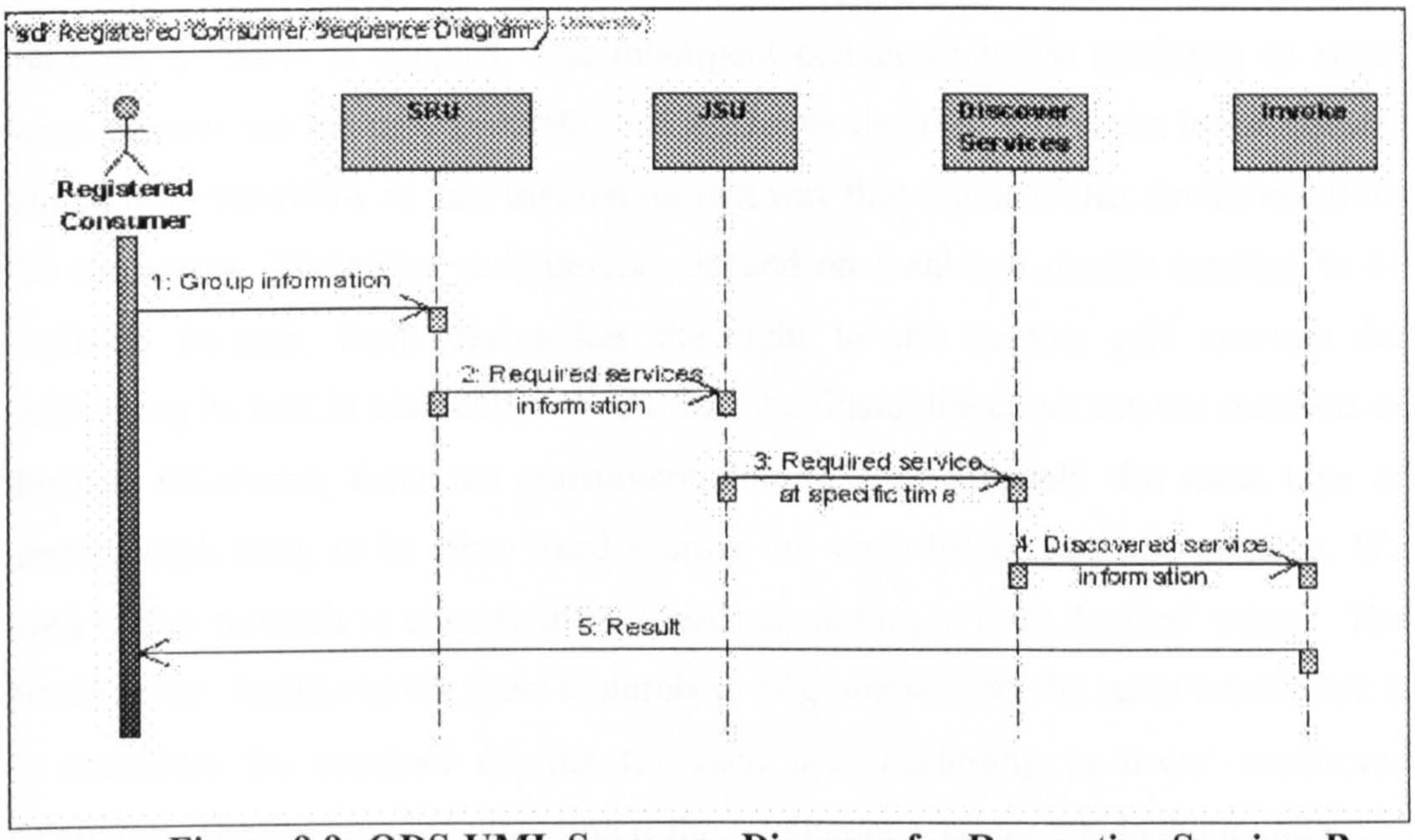


Figure 9.9: ODS-UML Sequence Diagram for Requesting Services By Registered Consumer

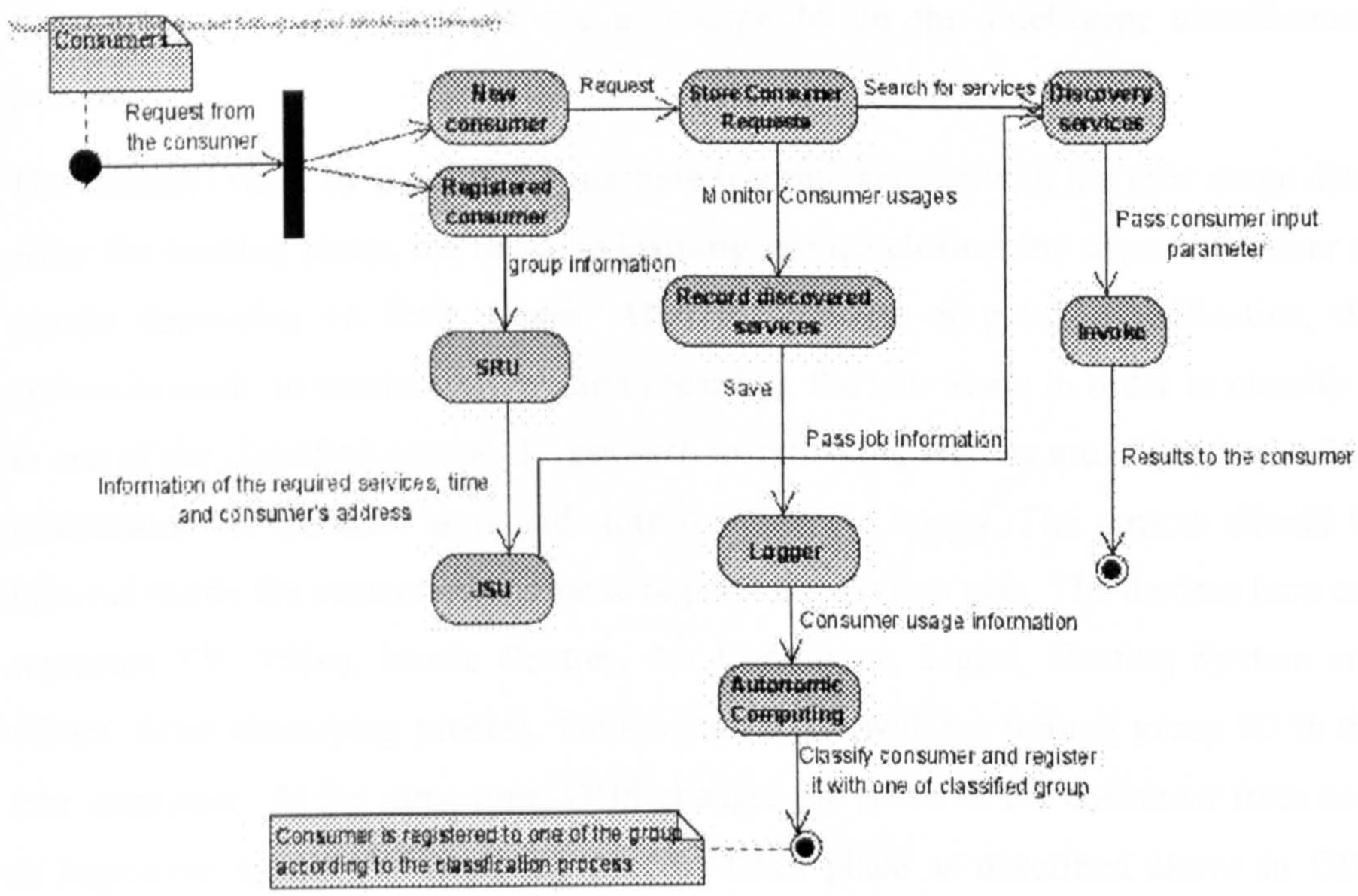


Figure 9.10: ODS-UML Activity Diagram

9.4.1.3. Case Study: On-Demand Service for Intelligent Connected-Home Machines

To elucidate the idea of ODS based on user usage classification, connected home machines scenario is adopted. The intelligent connected home machines or smart home devices are the next generation of the home devices. The smart home devices should have the ability to tune themselves in a way that offers a relax environment for the consumers. Therefore, such devices depend on local and remote services to be available on-time. Each device has the right to use remote grid services for performing its task in adequate way. On the other hand, the client can use numbers of devices. Of-course, there are consumers sharing approximately the same type of devices each time, or in other word sharing the same behaviours and patterns. We tried in this research to classify these clients according to their devices' usages. The benefit after classifying the users to numbers of group sharing the same behaviours is to anticipate the required devices for each user according to users' monitored behaviour. The autonomic computing is the intelligent services inside the middleware that is responsible for doing the automated stuff. Machine learning services inside the autonomic computing services are in charge to do the intelligent classification process.

The scenario starts by training the machine learning service with the user usage data. After the training phase, the machine learning service clusters the users to number of groups depending on their usages. After the process of group classification, the system is ready to receive information regarding the new users in order to classify it to one of the classified groups. To get such information, sensors are utilised to collect information for the new users and store it inside the logger. The sensors should be injected inside the consumer's home to register the devices uses. The devices here can represent TV, Video, Music Centres, Air Conditions, Lights, Heating System and others. After classifying process, the system replies with his joining group ID to the new consumer. At the same time, ODS changes the status of the consumer from new to registered consumer. Then, the process takes place as described above in ODS scenario. ASIDL is employed for deploying devices with assembly services and

infrastructures framework. Figure 9.11 shows an example of deploying devices with ASIDL.

```
<ServiceAssembly SA_ID="1">
    <Name>Research Application Service</Name>
    <Comments>an example of services assembly description language
</Comments>
    <Version>1</Version>
    <ServiceAggregation>1</ServiceAggregation>
    <ServiceAggregation>2</ServiceAggregation>
    <ServiceAggregation>3</ServiceAggregation>
    <InfrastructureAggregation>I1</InfrastructureAggregation>
    <InfrastructureAggregation>I2</InfrastructureAggregation>
    <InfrastructureAggregation>I3</InfrastructureAggregation>
    <Service ServiceID="1">
        <ServiceName>Light</ServiceName>
        <ServiceDescription>to control the lighting
         services</ServiceDescription>
        <ServiceCategory>Light</ServiceCategory>
        <ServiceType>Research</ServiceType>
        <Container>cmpwomar</Container>
        <ServiceLocation>cmsgris:8080/userdoc/Calculator
         </ServiceLocation>
        <InterfaceName>LightInterface</InterfaceName>
        <InterfaceLocation>http://cmpwomar:8080/userdoc/
         </InterfaceLocation>
        <Dependency>Antecedent </Dependency>
        <Authorized>True</Authorized>
        <AvailableLanguage>Eng,Fr,Ar</AvailableLanguage>
        <Be_Infrastrucre>True</Be_Infrastrucre>
         <Methods>
            <MethodName>TurnLight</MethodName>
            <InputParameters>ON,OFF</InputParameters>
            <InputParameterTypes>String,String </InputParameterTypes>
            <OutputParameters>Status</OutputParameters>
            <OutputParameterTypes>String</OutputParameterTypes>
        </Methods>
        <ContractInformation>
            <ContractID>1</ContractID>
            <ContractName>CMPWomar </ContractName>
            <ContractLease>1/1/2007</ContractLease>
          </ContractInformation>
           <RequiredInfrastructure />
    </Service>
</ServiceAssembly>
```

Figure 9.11: ASIDL Example for Deploying Home Device

Self-Organizing Map (SOM) is adopted in this case study to carry out the classification process. SOM is selected since it is one of the unsupervised learning techniques. Such technique is required in this case according to the absence of known target for the classification process. The data collected from the middleware repository has wealthy information to be processed, and starting with the definition of

SOM method as a vector quantization method, which places the prototype vectors on a regular low-dimensioned grid in an ordered fashion [140].

9.4.1.3.1. Generating Training Data for Connected-Home Application

An experimental data are required for the automated classification of users and their home devices' usage models. This data has been employed for training the SOM model to predict the classified group for connected-home machine. Simulation software has been developed and implemented using VS.Net and Matlab to generate training and testing data for SOM model.

Data format for SOM is a main concern. Therefore, we try to prepare the samples in proper iterations, and then it will be easy to construct them and build the data structure. Matlab *SOM-struct* using *som_data_struct* functions included with the SOM toolbox [140] are employed to build the data structure. Data pre-processing are needed, which can be simple linear transformations, normalization or logarithmic scaling especially when the divergence of ranges of data is too high. This is performed using *som_normalize* function. After that, the scaled data is used to feed the training system of the SOM.

Different type of users' usages categories are selected to represent the common features in utilising the home-machine categories. Each one of these category contains a number of home-machine devices or services. 0 and 1 are selected to represent the status of the devices as "OFF" or "ON" respectively. Figure 9.12 illustrates a screen shot for generating training data for connected home machine.

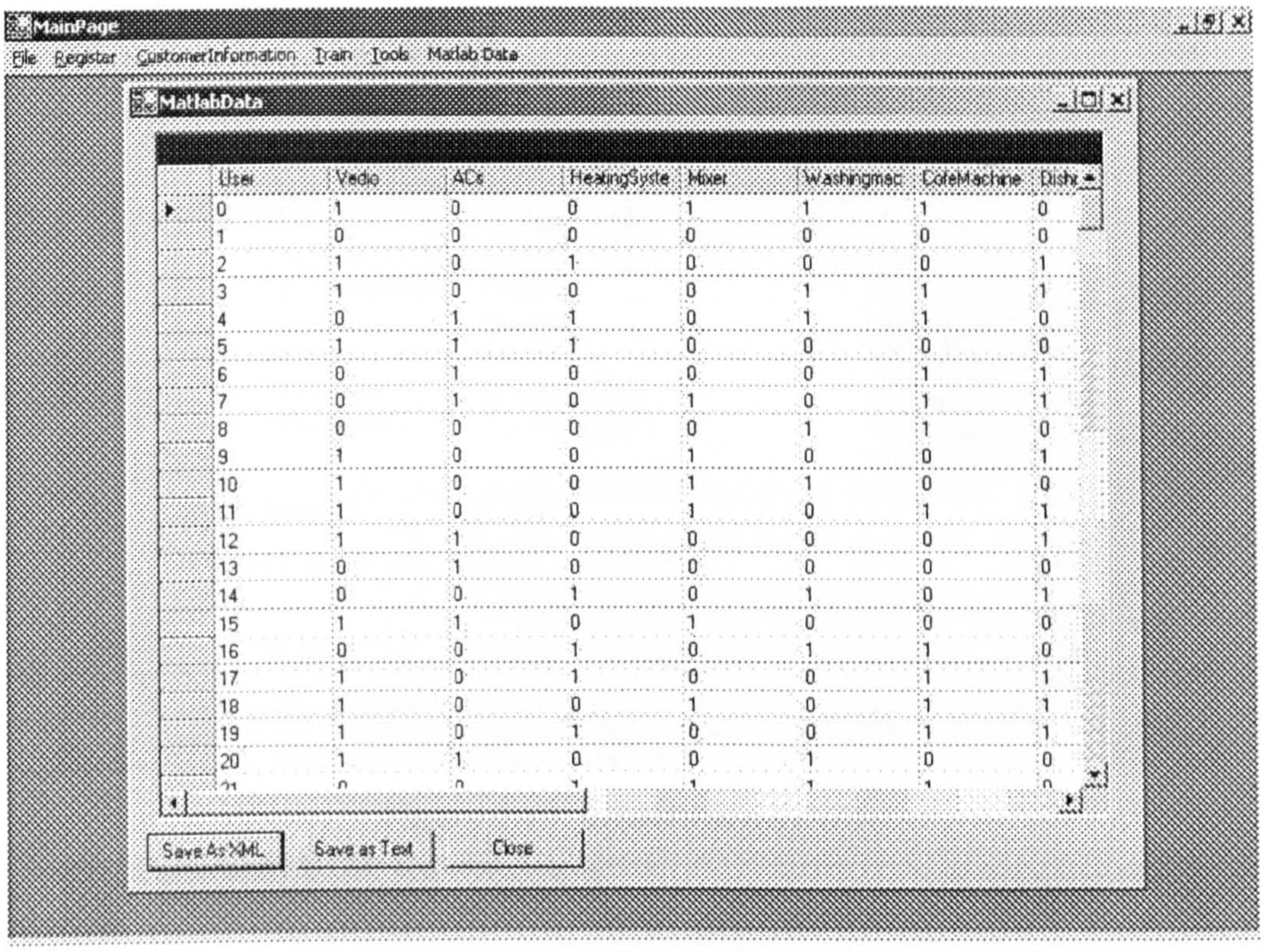


Figure 9.12: Screen Shot for Generating Home-Machine Training Data

The data generation software is developed to be able for generating any types of training and testing data in standard format depending on utilising XML. In this software, user can define the parameters and variables which are required to train the system. Figures 9.13 and 9.14 demonstrate the way of adding new groups and services, which in this case is related to connected-home machine. But it can apply for other types of system. Figures 9.16-9.19 present the process of generating of the training data. In this software, the user has the ability to select numbers of training users, and the way of generating the data, i.e.: the consumer uses all devices or part of them. Also, the user has the right to select the parameters that can appear in the training data as shown in Figure 9.17. Then, the user specifies the path to store the training data, and the format of the data, i.e. row format or column format, as illustrated in Figure 9.18. The generated training data is shown in Figure 9.19.

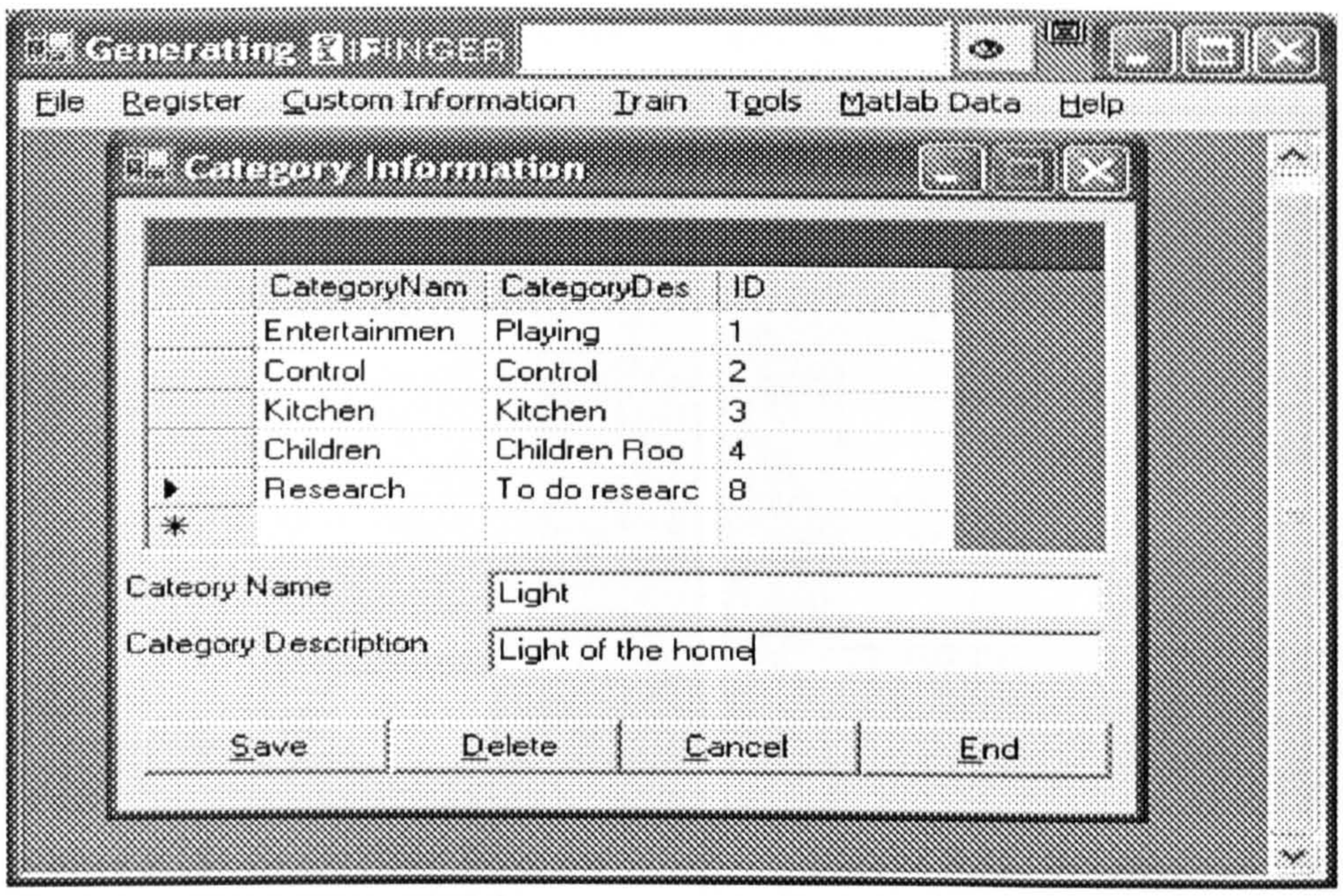


Figure 9.13: Generating Groups

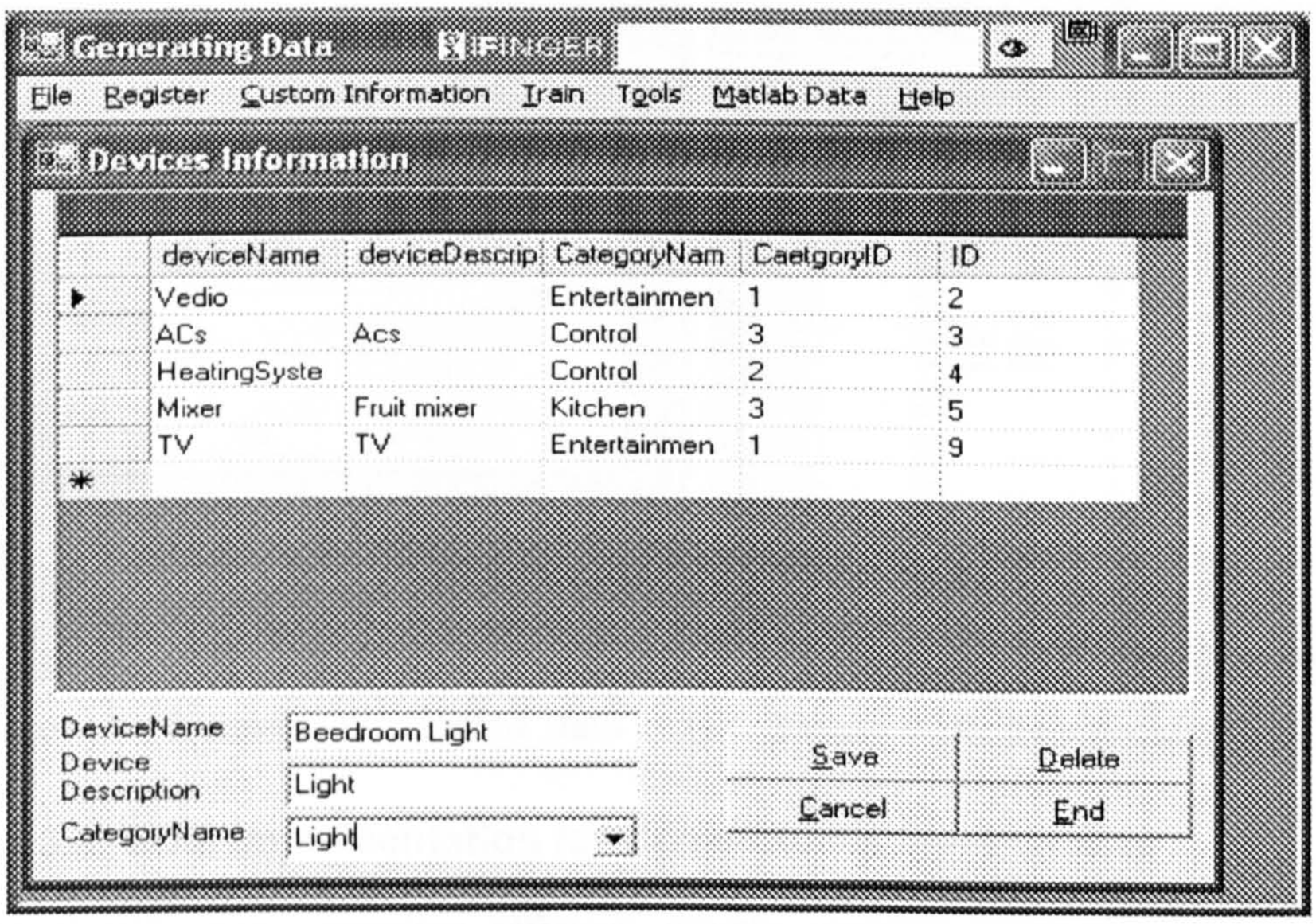


Figure 9.14: Generating Devices

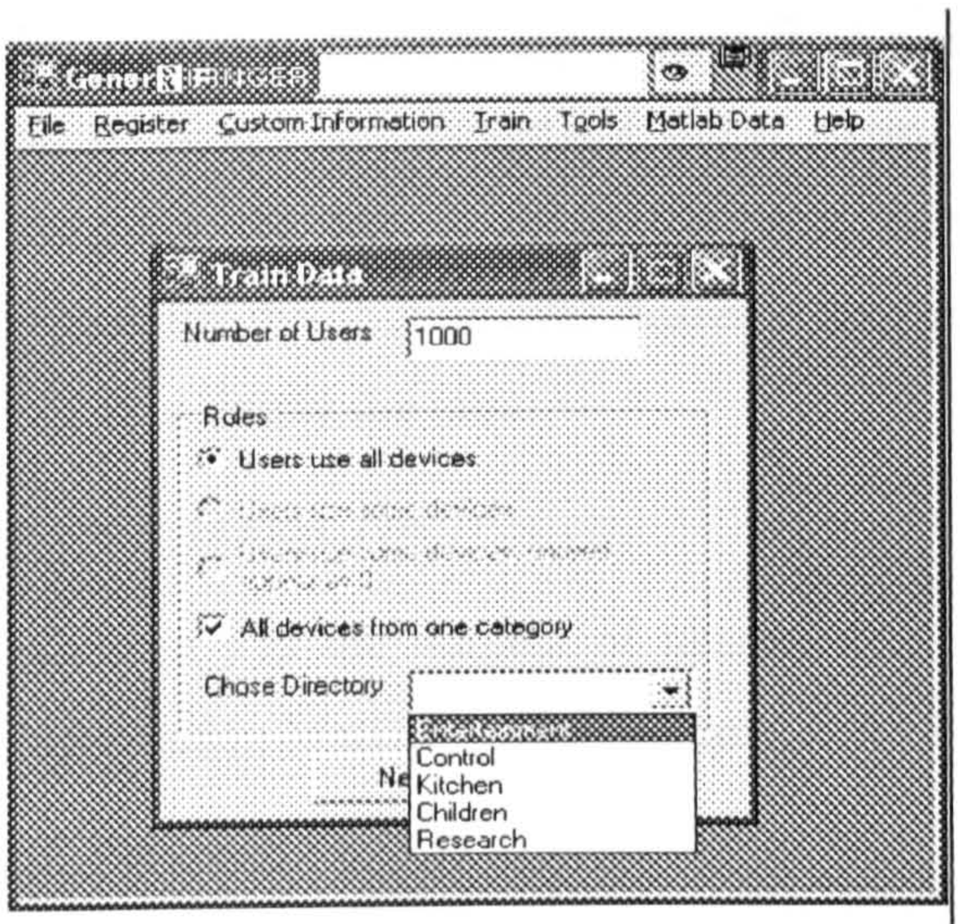


Figure 9.15: Generating Data for Training SOM With 1000 Users.

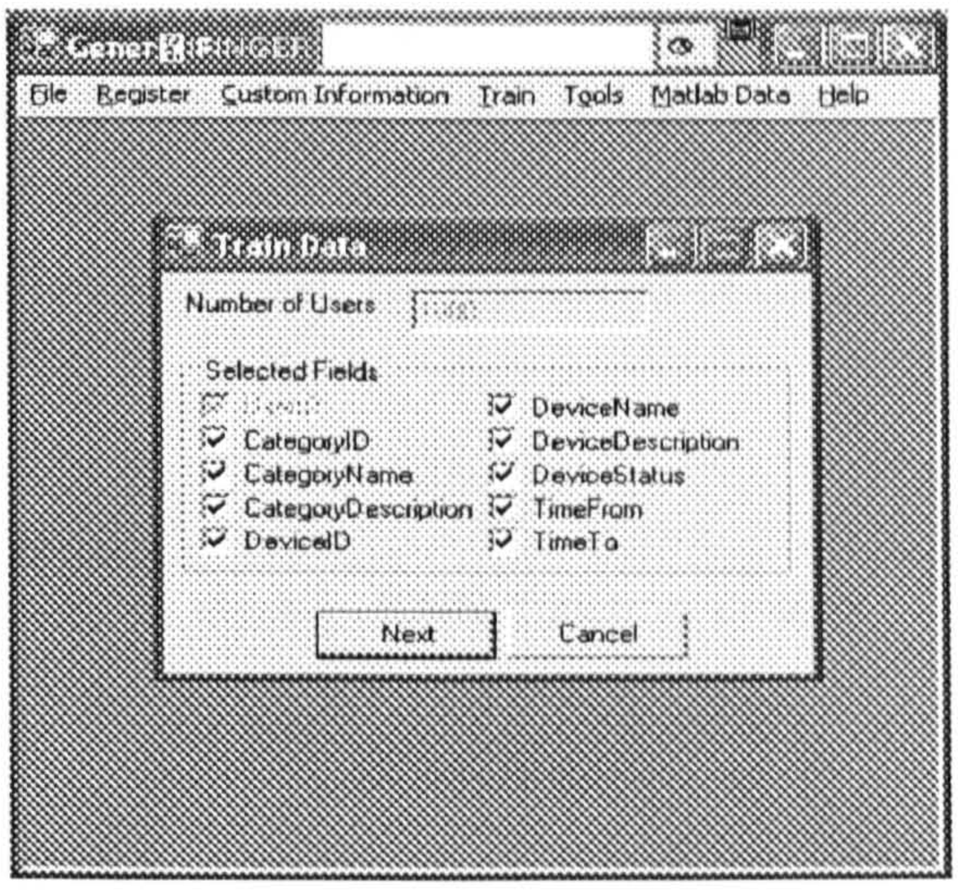


Figure 9.16: Parameters That is Required to be Added in the Model

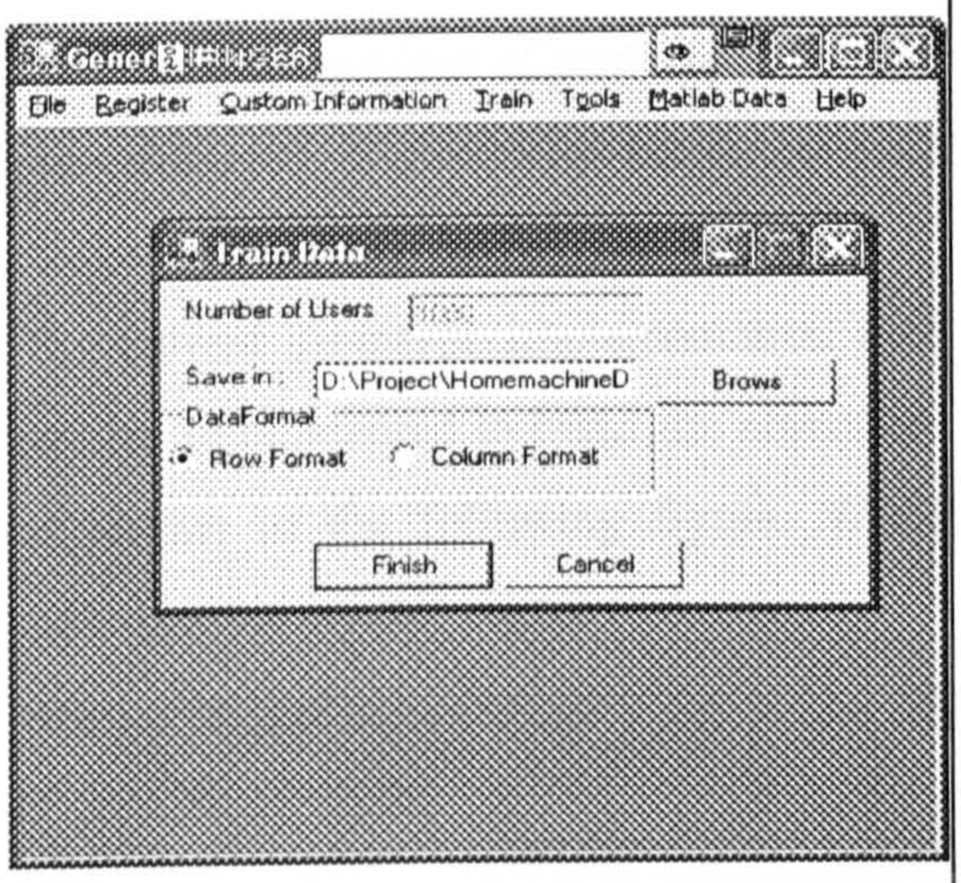


Figure 9.17: Saving Generated Data

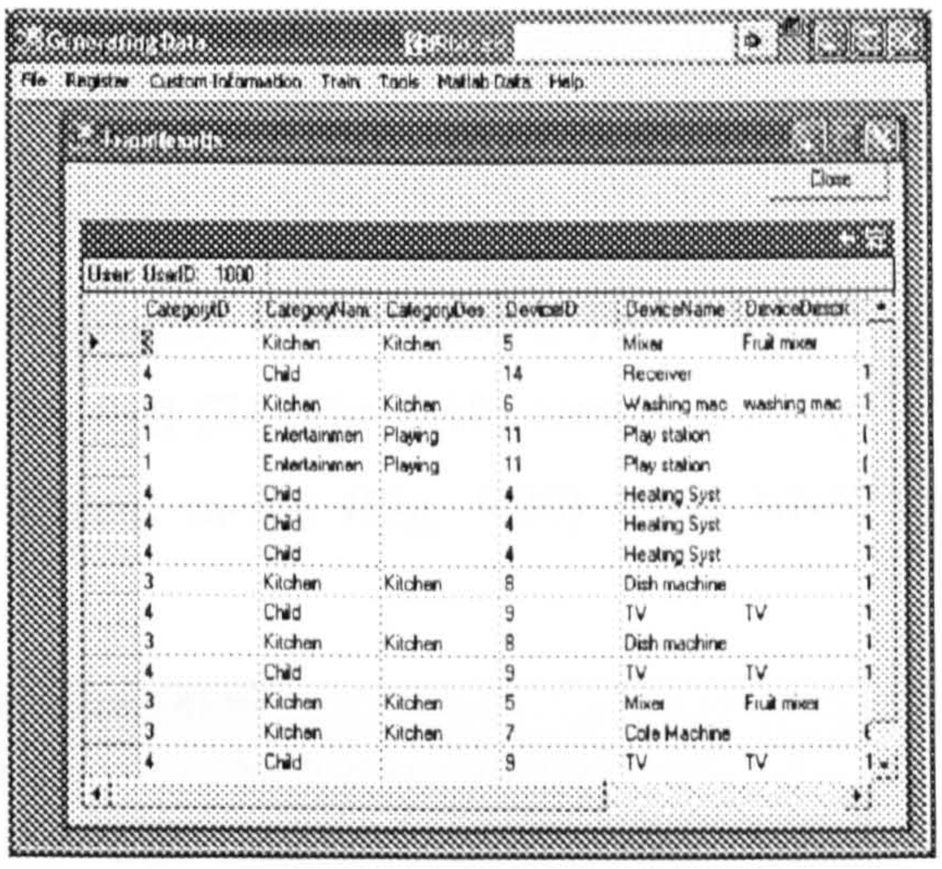


Figure 9.18: Edit Generating Data

9.4.1.3.2. SOM Implementation for Connected-Home Machine

SOM classification comprises efficient, mostly accurate mean as a visual data analysis for the maps, while in our approach we have to reduce the role of administration to the minimum. Thus we headed to use other classification algorithms like K-Nearest Neighbourhood (KNN) [155], which is suitable for similar cases, a special MatLab function would implement KNN classifier using arbitrary distance matrix. The asymptotic performance of the KNN rule is good and the rule is simple.

Given a set of labelled training examples, T, and an unseen test point, x, computing the squared distance in input space from x to each of the examples in T. Discarding all, training cases except for the K cases which are closest to the test point are assigning the test point to the most numerous classes amongst this KNN. A probability distribution over classes is also easily constructed. The probability of class i (p_i) is simply the ratio:

p_i = n_i / sum_j (n_j)

Where n_j is the number of nearest neighbours in class j. KNN function is also supported in MatLab SOM Toolbox:

[C,P]=knn(d,Cp,[K])

Where 'C' is a matrix of size N*K of integers indicating the class decision for data items according to the KNN rule for each K. 'P' is a matrix of size N*k*K, which represents the number of prototypes for each classifier. 'd' is an N*P matrix which is pre-calculated dissimilarity (distance matrix). 'Cp' is a Px1 vector that contains integer class labels, in our case (OMAR, …, TALEB)

The current prototype implementation uses the Matlab SOM toolbox. Two MatLab m-files implemented to build the classifier. The first file (Fig. 9.19) creates the map as shown in the following figure. While second m-file (Fig. 9.20) build KNN classifier for input P vector

```
%% This script will create a function to generate SOM map from raw data file.
% Then saves the trained map into input file, which has to comply to the SOM_PAK formal/format
% Output file should have .mat extension.
function som_make_map(inputfile,outputfile)
% The algorithm/procedure: achieved in a number of steps;
% 1. load and read input data samples
sD=som_read_data(inputfile);
% 2. we will check for error if no errors are found then do action 1 or else do action 2
% 3. We normalize the data
sD=som_normalize(sD);
% 4. we create the SOM map and train it using auto training function som_make.
sM=som_make(sD);
% 5. We shall denormalize the data map
sD=som_denormalize(sD);
```

```
% 6. we shall save the trained map into outputfile
som_write_data(sD,outputfile);
% End
end;
```

Figure 9.19: Implementation of the Map Function

```
%% This function will crate a knn nearest neighbourhood classifier for a given input
% trained SOM map then find estimated best matching class for an input P vector
% in addition to adapt the current map with the input vector P then returns class
function myClass=som_classify(inputfile,P);
% First we shall read the trained map file
sD=som_read_data(inputfile);
% Then we will check for no error
% if no error then .... else
% Then we shall check input vector P for dimensionality matching with out map
% Check for vector missing data NaN's
% if NaN(P) ... then ... else....
% Get the integer class labels for prototype vectors
[Cp,label]=som_label2num(sD);
% Calculate euclidean distance matrix
d=som_eucdist2(P,sD);
% Classify using 1,2,3...,10-rules, be careful of different knn.m function name in neural net path
class=knn(d,Cp,10);
% includes results for 5NN
class(:,5);
% Original class labels for 5NN
myClass=label(class(:,5))
% save adapted SOM map
som_write(sD,inputfile);
% End
end;
```

Figure 9.20: Implementation of KNN Classifier for a Given 'P' Vector

The output of *som_classify.m* file is a label class, which after training the map represents the nearest match for the input data *P* vector. Following our experiment it can be seen that from 16 features, 10 classes with 1000 training epochs, the resulted KNN classifier succeeded to achieve 7 out of 10 *P* input vectors.

9.4.1.3.3. Results of SOM Classification for Connected Home Machine

The results of the experiments are obtained using an implemented machine learning middleware service. Matlab SOM library [140] is utilised to implement such

experiment. Figures 9.21, 9.22, 9.23 and 9.24 show SOM-based classification results of the data set generated from Matlab, which represent a simulation of our self-managing middleware for intelligent connected-home networks. The results represent classification for different types of users and devices. Figure 9.21, shows many correlations between devices, which are obtained after the training phase which included 1000 input sample data and 10 trainees. As shown in Figure 9.21, sample of these correlations are described in the following points:

- Lights and PlayStationII correlation
- Video and Coffee Machine correlation
- Video CD and Fans correlation
- Vacuum cleaner and Washing machine correlation

Figure 9.22 represents U-Matrix distribution of labels for the connected home devices. Figure 9.23 shows shaped U-Matrix with coloured regions exhibiting three clear clusters of the map. Figure 9.24 demonstrates probability distribution function (PDF) of the input vectors. The critical analysis of this approach depends on selecting and scaling the correct data for training the system, because using un-normalised data might produce inaccurate user classification. Therefore, selecting the adequate training data is the vital to get right classified model.

At runtime, the machine learning middleware service, along with the training data (user and device classification) can classify logged users according to known users (one of the seven classified regions). These classes specify the user types and their usage model, such as the device usage order and time of usage. This is used in this case study to guide the autonomic middleware services for service reservation and provisioning.

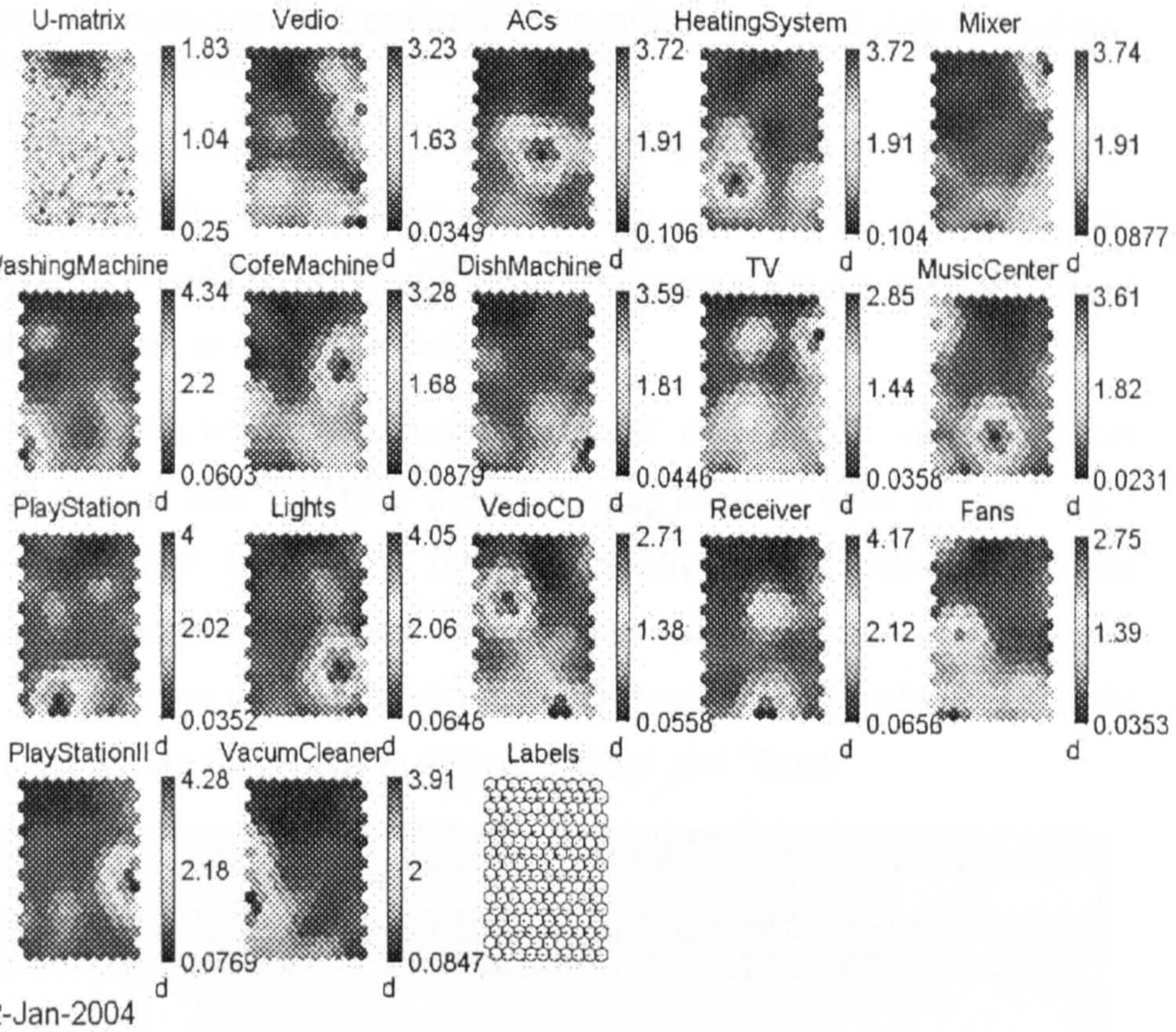


Figure 9.21: SOM Visual Classification

Figure 9.22: U-Matrix Distribution of Labels

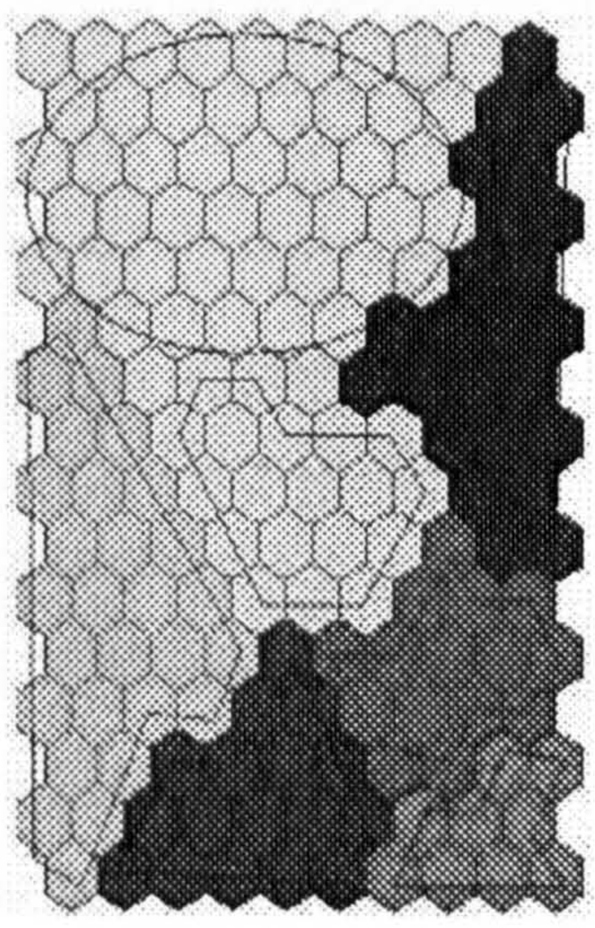


Figure 9.23: U-Map of SOM Maps Resulted Data

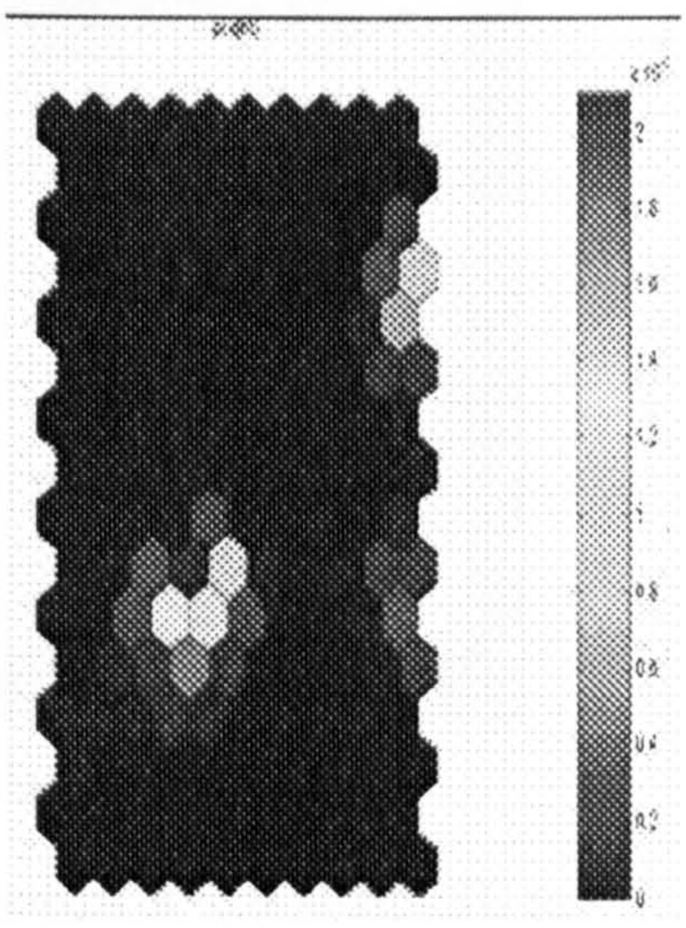


Figure 9.24: Probability Distribution Function PDF of the Input Vectors

9.4.1.3.4. Implementation of Service Reservation Unit and Job Schedule Unit for Connected-Home Machine

To this end, autonomic computing based on SOM is utilised for classifying the groups of users according to the shared behaviour, which is expressed by devices' usages. The new consumer is codified to one of the classified groups and his status is changed to registered consumer, as mentioned before.

In this case study, two registered consumers ('Wael' and 'Taleb') request services from the ODS system. They send their classified group ID to the SRU. In which, SRU requests the behaviour information for the classified group. This information indicates the required resources, time of operations and other related information. Figure 9.25 demonstrates a list of devices required by user 'Wael'. Also it shows the two consumers who use the system, in this case 'Wael' and 'Taleb'.

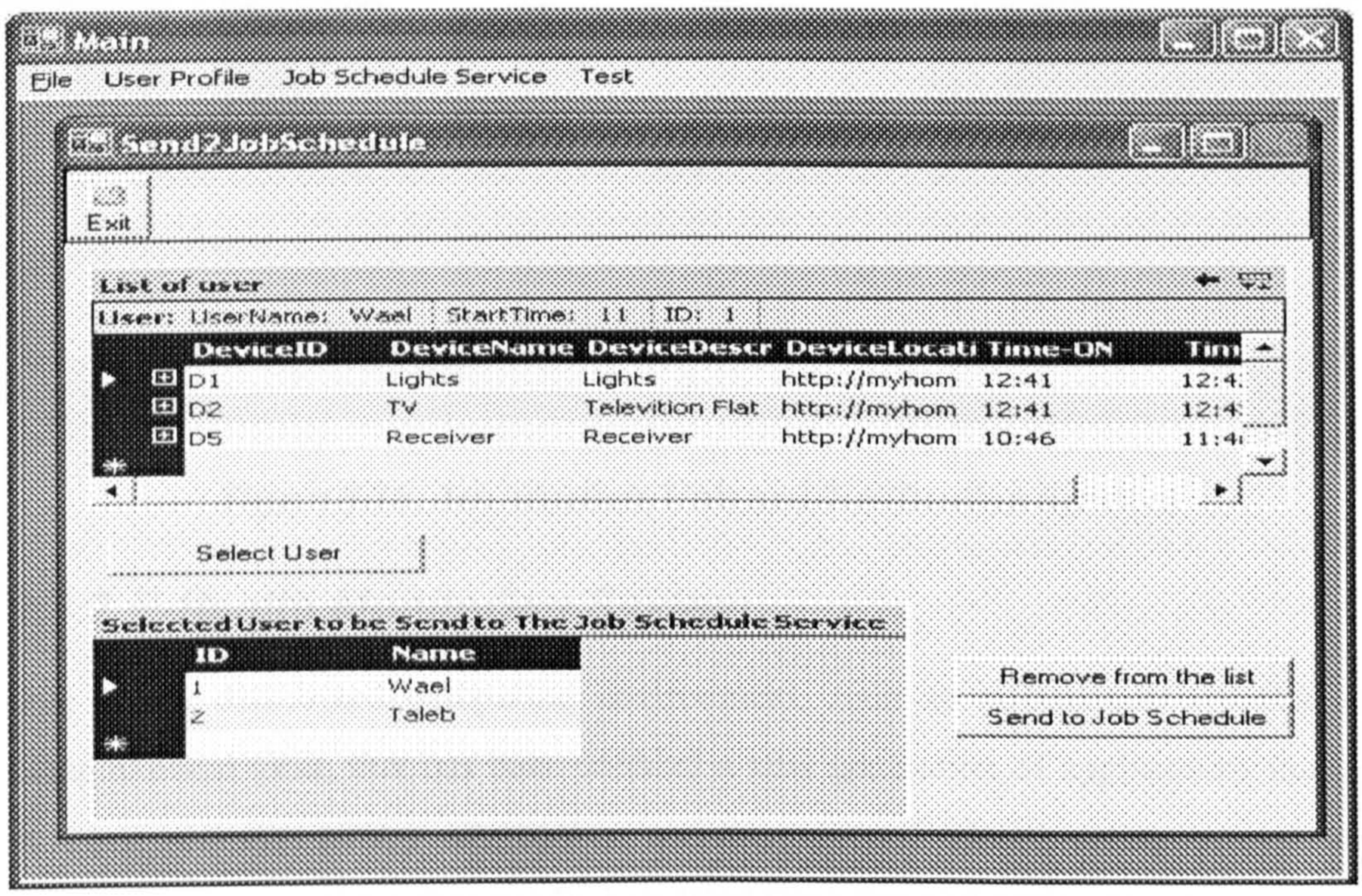


Figure 9.25: Service Reservation Unit

This information regarding the new users with their requirements is sent to the JSU to manage the execution of such services. For instance, Figure 9.26 presents the notification of execution service (device) light for user 'Wael'. It also describes the time of execution for this user. The system predicts the time for the execution for

each service based on the prior collected information. Such information is collected by injecting on-fly sensors. Of course, the user has the right to change the time of service execution or even the service in order to tune the system and make it suitable for his/her or it needs.

The software for the SRU, notification system and JSU for connected home machine is implemented using VS.Net. Figures 9.25 and 9.26 illustrate screen shots of such system.

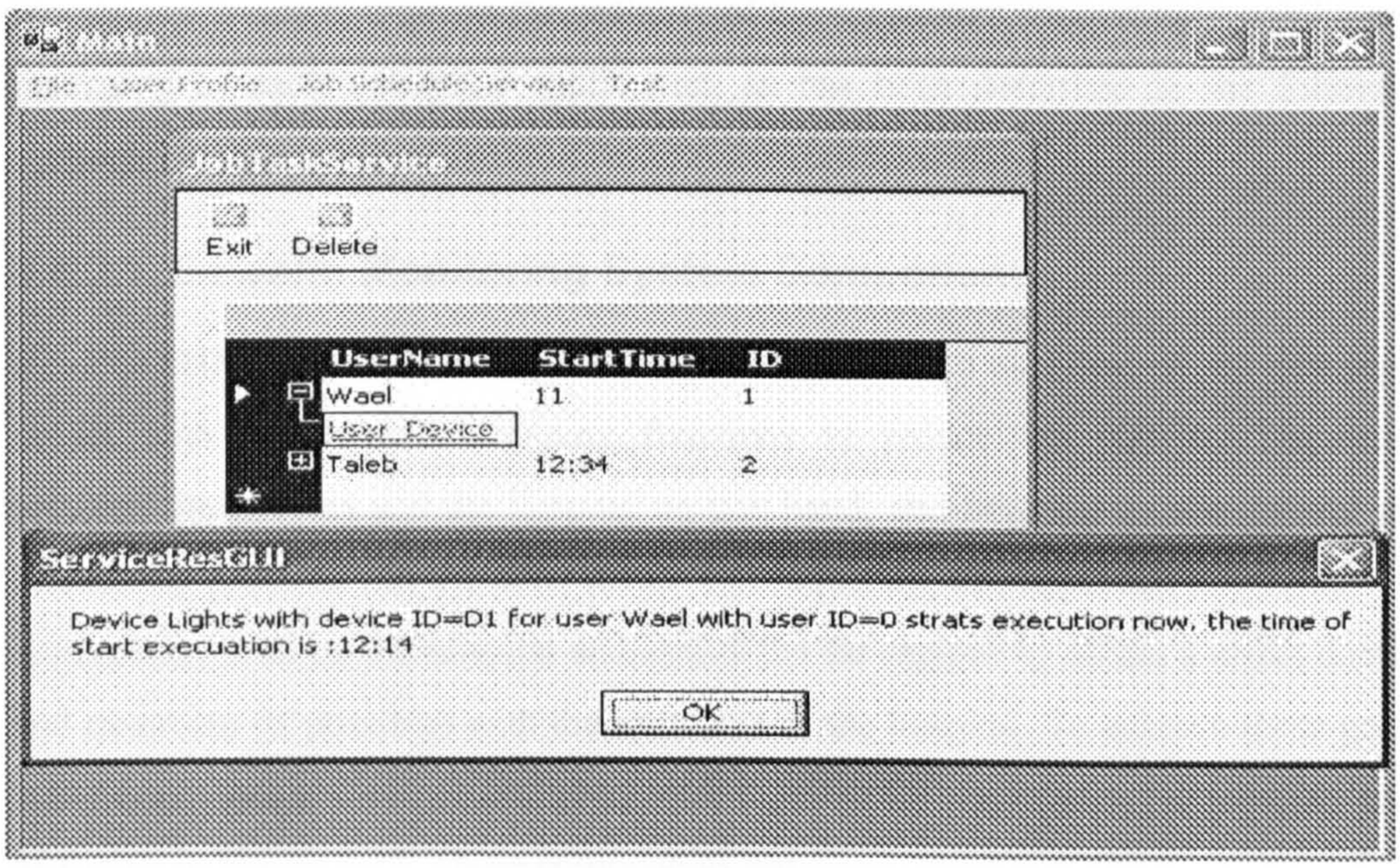


Figure 9.26: Job schedule Unit and Notification Services

9.4.2. E-Health Monitoring System

Due to the distributed computing environment, such as information systems and computational Grid, that has been enabled a new generation of applications that are based on seamless access, aggregation and interaction. E-Health enterprise system is one of such applications that can be benefited from the grid computing overlay. Grid computing infrastructures, utilities and services can improve efficiency, effectiveness, access and quality of clinical healthcare industry along with the reduction of the cost of ownership. This section presents a way for implementing remote patient monitoring system based on grid computing overlay. Self-management system is

embedded in the E-Heath Monitoring System (EHMS) depending on the self-configuration, self-prediction and self protective concepts.

SAF with its description language (SADL) are adopted in this design for deploying variety of medical sensor along with other grid computing sensors and actuators. Monitor Session Description Language (MSDL) is employed in this scenario for offering a semantic way to exchanging information between the requester (hospital) and the e-health framework. Case study for monitoring the situation of the pregnant women is developed to prove the usability of the autonomic computing capabilities in managing e-health enterprise application, as has been described in the following sections.

9.4.2.1. E-Health Monitoring System Components

Remote EHMS is proposed in this research to show the usability of the developed model with autonomic computing functions to reduce the physical interaction between the patients and the hospital on one hand, and improve remote control and monitoring processes on the other hand. Consequently, this approach abbreviates the unnecessary load on the hospital and assists the far regions to obtain a direct contact and treatment (if possible) with the specialist in the hospital. To achieve these goals, many components are necessitated to work together for performing the required tasks as shown in Figure 9.27. These components are:

- **Monitoring System:** to manage the process of gathering readings from the patients and save them in the patient profile inside the logger depending on semantic format, such as XML. The monitoring system for E-Health environment consists of number of health sensor (temp, pressure, blood, etc...), analyzer and actuator, logger and schedule services. Monitoring system runs inside the middleware. The monitoring system model has been explained in Section 7.4.
- **Pregnancy Health Monitoring System:** gets the reading from the monitoring system in open standard format. Then, it manages these readings as specified by the medical schema which define the rule for analyzing the readings, like

normal blood pressure. Such schema is provided by the hospital or medical system.

- **Autonomic Computing Service:** performs number of intelligent stuffs. The first responsibility is derived from the self-protective capability, which is expressed by checking the contract of the requesters. Self-organising capability is the second responsibility of the autonomic computing. Where, autonomic computing discovers the best appropriate resources from sensor and services from resources container, which achieves the requesters' demands. Self-optimising capability is also included in autonomic computing responsibilities. Classifying and predicting the personification of the monitored cases is a paradigm of self-tuning concept.
- **Alert service:** generates and deliver the alert messages to the hospitals and patients. The alert messages are generated from the predicted personification.
- **Planetary Scale System:** offers the domain for performing such large-scale and enterprise system. Grid computing is adopted in this scenario according to its high availability and reliability in utilizing resources from hardware, networking, autonomic computing and sensors. This system is proposed to be built depending on employing grid computing overlay.
- **Hospital or e-health applications:** represents the consumer in this system. It is responsible for two main tasks. The first one is generating a request for starting gathering information from the targets (patients). And the second one is to generate and provide the medical schema that is utilized for predicting the case.
- **Patient:** represents the monitored target in this case.

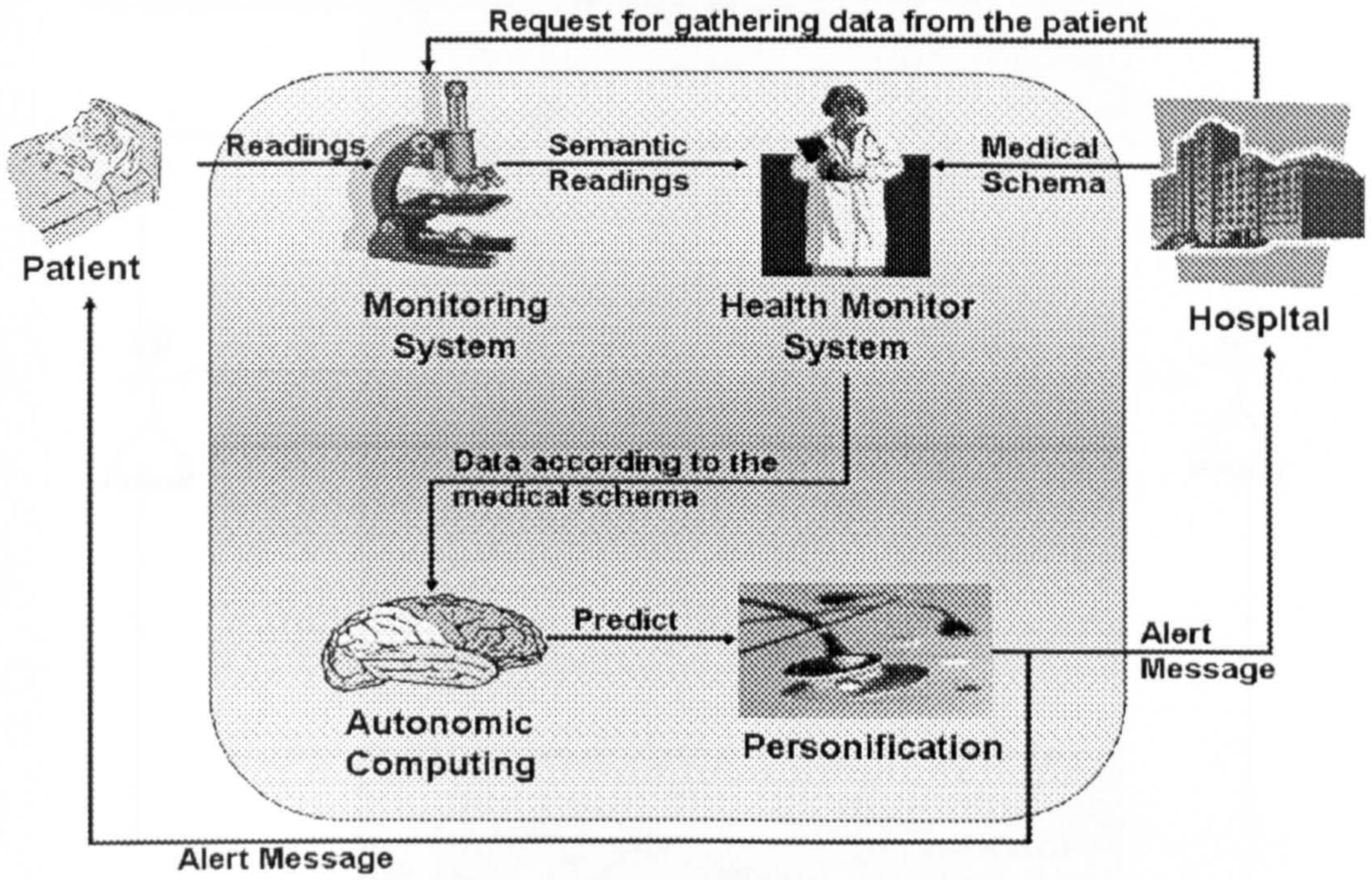


Figure 9.27: E-Health Monitoring System (EHMS)

9.4.2.2. EHMS Model

EHMS model consists of two players namely; *patient* and *hospital* as shown in Figure 9.28. The patient represents the monitored target of the system, while the hospital symbolized the auditing, control and requester of the system. Moreover, the model is composed of seven use cases namely; *Monitor System*, *Health Monitor System*, *Medical Schema*, *Autonomic Computing*, *Personification*, *Alert Service*, and *Resources*. Middleware represents the fabric for running monitor system, resources, and autonomic computing services.

Resources in this model deal with all types of services and infrastructures required for executing the expected jobs from auditing, predicting, storing and accessing information and others. The resources cover all types of general sensors, network sensor, medical sensor, actuator, logger, intelligent services for autonomic computing, and communication and networking services.

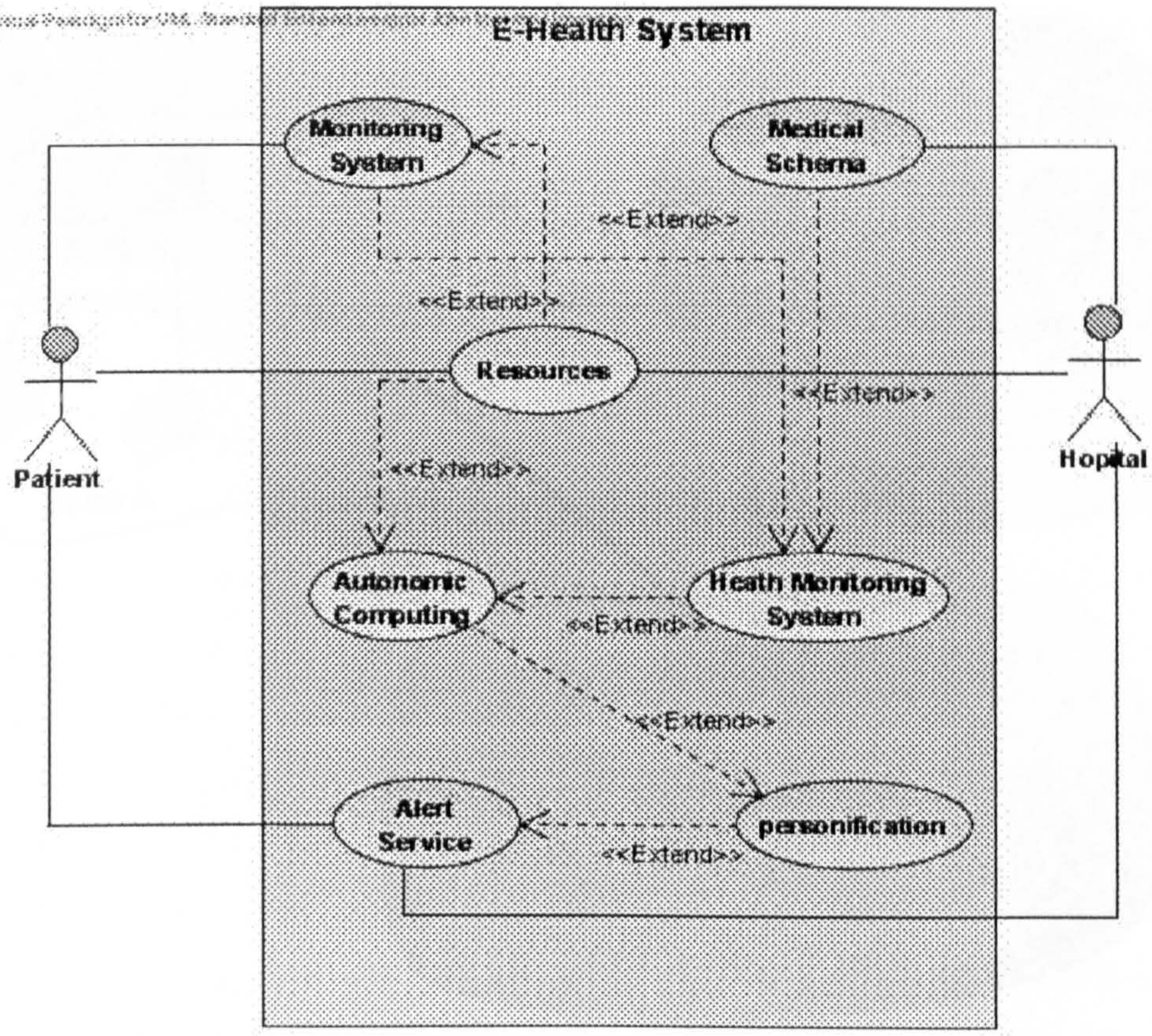


Figure 9.28: EHMS UML Use Case Diagram

The monitoring system, which is different from health monitoring system, is in charge of collecting information from the patients according to the hospital's requests. Moreover, it is in charge of delivering the data after analysing process in semantic format to the health monitoring system.

Monitoring system liaise information with other monitoring system in another zone/cloud to achieve the ultimate optimisation and increasing of the QoS and reliability in the process of collecting data from the patients. Sensor manager agent of one zone or cloud, which was described in chapter 5, is responsible for interacting with other sensor manger agents in other zones. Each zone can be presented as hospitals with its patients as shown in Figure 9.29.

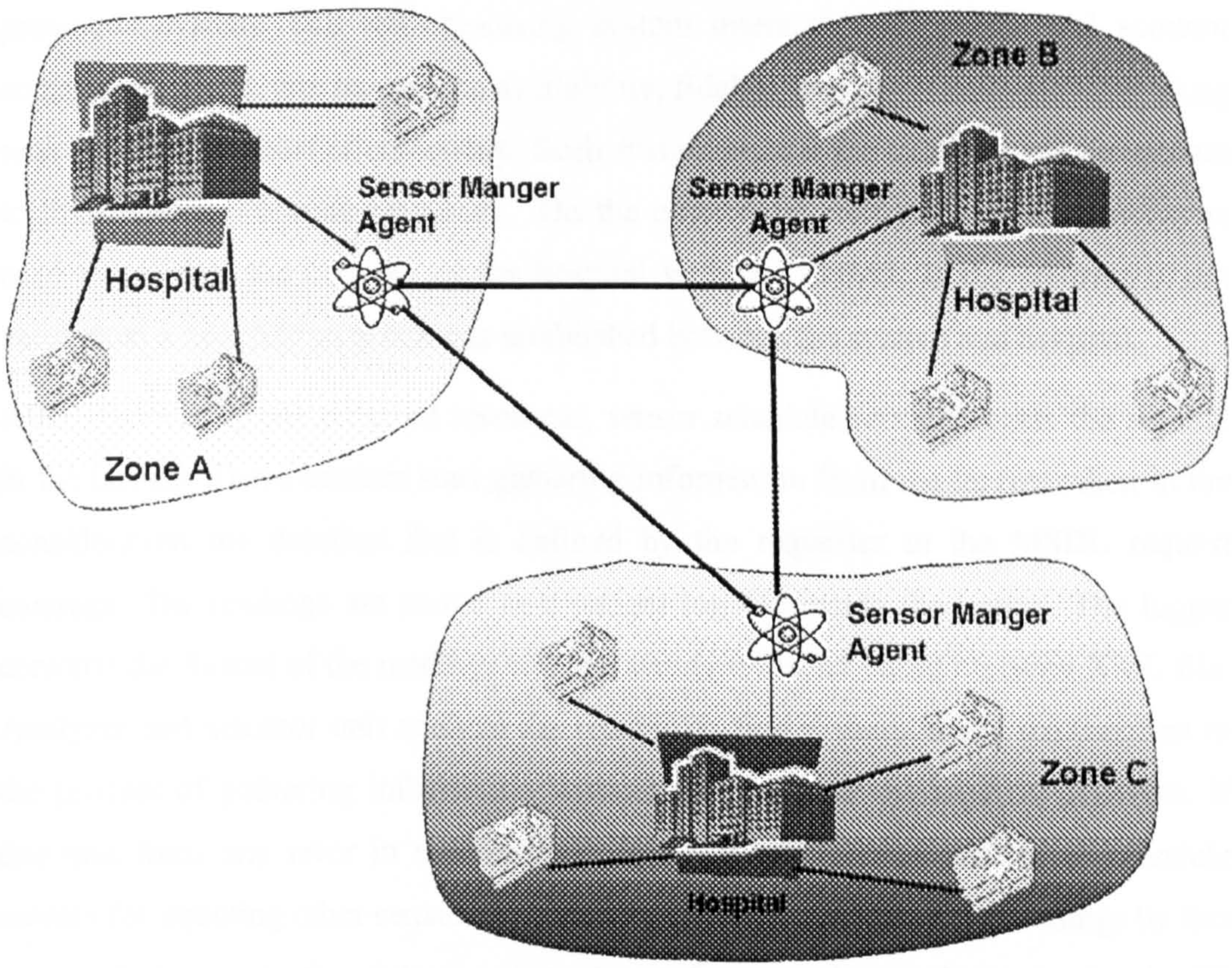


Figure 9.29: Zones for Health Monitoring System

9.4.2.3. E-Health Monitoring System Scenario

The scenario of the remote e-health monitoring system is commenced by generating a request by the hospital in order to monitor patient, as shown in the UML sequence and activity diagrams in Figures 9.30 and 9.31. This request includes information regarding the required sensors, target information, and authority. This request should be in standard format to be acceptable in all types of system. For this reason, MSDL is adopted to be used for dispatch a request from hospital to the EHMS. MSDL has been described in details in chapter 8.

The MSDL request is received by the sensor schedule service inside the e-health monitoring system. This unit analyses the requirements from the MSDL message. These requirements cover the type of duration, targets, required resources, and contract. In order to find the most appropriate resources, sensor schedule service incorporates with autonomic computing service to offer self-organising and self-

protective system. The self-organising system interacts with sensor and actuator container in a way that insures the availability, fidelity, high QoS and reliability along with reducing the cost of ownership. Such system insures the capability of the system to find most appropriate resources. On the other hand, self-protective capability is used to validate the contract of the hospital with the EHMS. If the contract is not valid, then a negotiation process is established between the system and hospital.

After discovering the required resources, sensor schedule service injects the sensors in the targets. These sensors start gathering information from the targets taken in the consideration the duration that is defined by the requester in the MSDL request message. The readings are stored in a patient log file inside the logger. The logger converts the format of the readings to open standard format based on using XML file. Analyzer and actuator unit analyze the reading to find if there is any malfunction in the process of gathering information without any touch to the medical concepts. If this unit finds any error in the readings, it sends a request to the sensor schedule service for injecting other sensors into the targets. After analyzing the readings by this unit and discovers the data is adequate, it forward the readings to the health monitoring system. And by this process, the job of monitoring unit is completed.

Health-monitoring system receives messages from two sources. The first source is the analyzer and actuator unit in the monitoring system, as described in the previous paragraph. The second message is received from the hospital which includes the medical schema. Such medical schema is utilized to describe the medical concepts that are used in understanding and analyzing the collecting readings, as an example the blood pressure and pregnant schemas. Autonomic computing service is coming to the light for carrying out the prediction process. Autonomic computing generates the personification for the monitored patient depending on the sensors' readings and medical schema. This personification is sent to the hospital and patient depending on the alert message service. The deliverer alert message indicates if the case is dangerous, critical or ok.

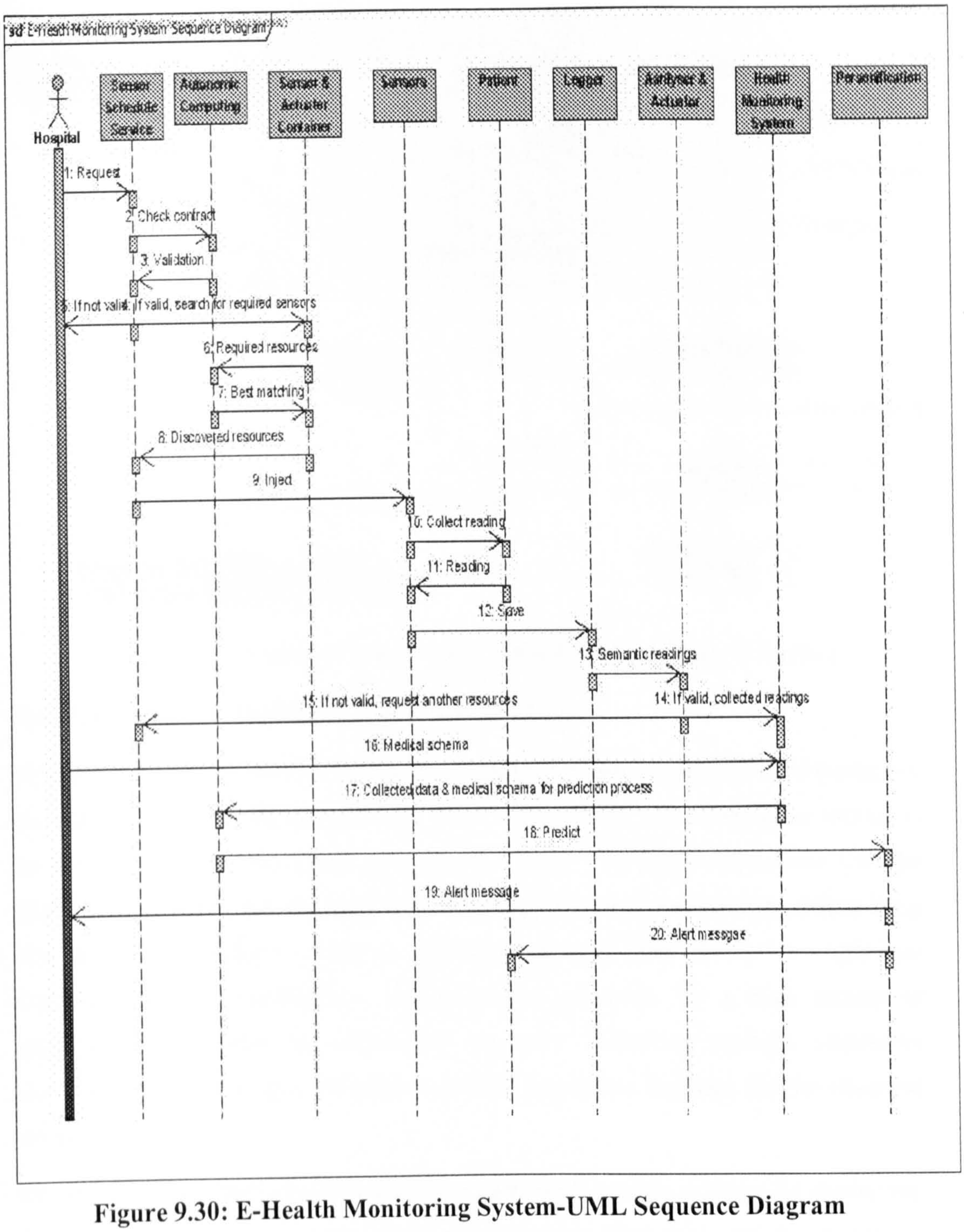


Figure 9.30: E-Health Monitoring System-UML Sequence Diagram

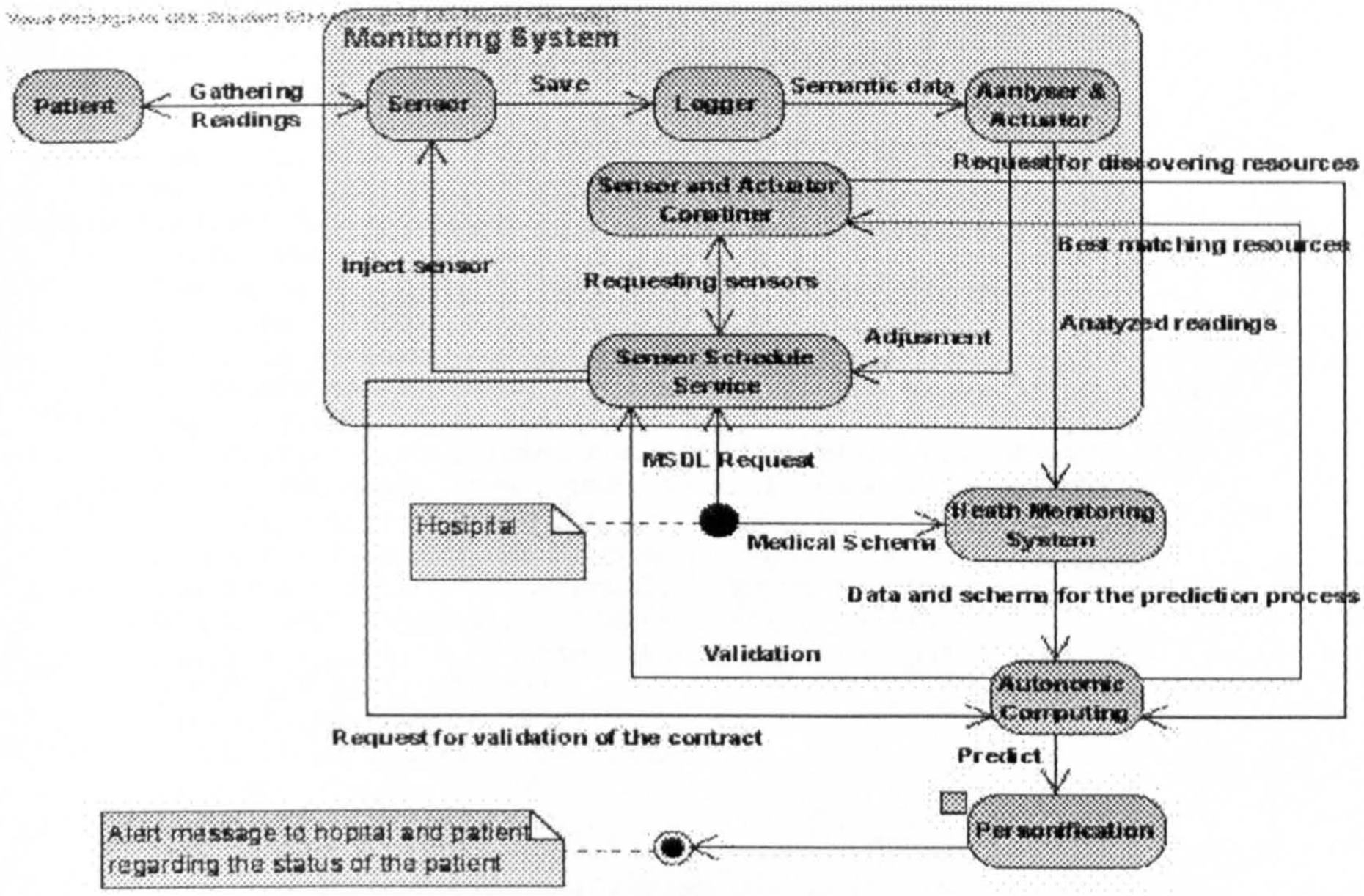


Figure 9.31: E-Health Monitoring System-UML Activity Diagram

9.4.2.4. Prediction Service as Web Service

Prediction service is the intelligent part of the system that is responsible for predicting the personification from the gathered data at the patient side. Prediction service is running inside the autonomic computing service. Different algorithms can be employed to implement the prediction process, like machine learning, data mining and analysis algorithms. Analysis algorithms proved their ability to implement different types of classification and prediction processes for a huge variety of applications depending on mathematic analysis. Therefore, multiple regression analysis is adopted in this scenario to predict the personifications for the observed patients.

As mention previously, grid computing is employed to offer a fabric for deploying, discovering, and invoking services and infrastructures. Therefore, multiple regression analysis is developed and implemented as web service to be integrated with the services of the middleware. The mathematical model for multiple regression is shown in Sec. 7.2.2. Figure 7 demonstrates VB.Net code for implementing multiple regression as web service.

```
<System.Web.Services.WebService(Namespace :=
"http://tempuri.org/E_Health_Sys/MultipleRegression1")> _
Public Class MultipleRegression1
    Inherits System.Web.Services.WebService
'Web Method Section
    <WebMethod()> _
Public Function MultipleRegression(ByVal path As String) As Double()
        'Path: Path of the data
        Dim ds As New DataSet 'for storing data in dataset
        Dim n As Integer 'Number of raw data
        Dim k As Integer 'number of Variables
        ds.ReadXml(path) 'Read the data as XML and store it in dataset
        Dim i, j, i1, j1 As Integer
        k = ds.Tables(0).Columns.Count 'To store number of variables
        n = ds.Tables(0).Rows.Count 'To store number of raw data
        Dim D(n - 1, k - 1) As Double
        Dim X(n - 1, k - 1) As Double   'Input matrix (X)
        Dim XT(k - 1, n - 1) As Double  'Input transpose matrix (XT)
        Dim XInv(,) As Double   'Input inverse matrix
        Dim Y(n - 1, 0) As Double 'The training output
        Dim M1(,) As Double 'X*XT
        Dim M2(,) As Double 'XT*Y
        Dim b(,) As Double  'The coffecient
        i1 = 0
        Dim dr As DataRow
        Dim dc As DataColumn
        For Each dr In ds.Tables(0).Rows
            j1 = 0
            For Each dc In ds.Tables(0).Columns
                If j1 = 0 Then
                    X(i1, j1) = 1
                    XT(j1, i1) = 1
                Else
                    X(i1, j1) = dr.Item(dc)
                    XT(j1, i1) = dr.Item(dc)
                End If
                j1 = j1 + 1
            Next
            Y(i1, 0) = dr.Item(0)
            i1 = i1 + 1
        Next
        M1 = MatLib.Multiply(XT, X)
        XInv = MatLib.Inv(M1)
        M2 = MatLib.Multiply(XT, Y)
        b = MatLib.Multiply(XInv, M2)
        Dim b1(k - 1) As Double
        For i = 0 To k - 1
            b1(i) = b(i, 0)
        Next
        Return b1
    End Function
End Class
```

Figure 9.32: VB.Net Code for Multiple Regression Web Service

9.4.2.5. Case Study: Monitoring Pregnant Women

The technique of EHMS based on using grid computing overlay is applied for monitoring pregnancy status as a case study of applying such technique. The required tests for the pregnancy case are categorized into four major categories according to the interval of the pregnancy. This information regarding the required tests is adopted according to the standard test specification available at the local hospitals, which have been approved by the international test standards. Table 9.4 demonstrates the required pregnancy tests according to the intervals of the pregnancy.

Table 9.4: Pregnancy Tests

Pre-Conception	First Trimester	Second Trimester	Third Trimester
Weight	Weight	Weight	Weight
Rubella	hCG	Rubella	Rubella
Hepatitis_B	Gonorrhea	CVS	Urinalysis
Hemoglobin	Chlamydia	AFP-Maternal	HIV Antibody
Sickle Cell	Syphilis	hCG	Group B Streptococcus
CF_Gene Mutation	Urinalysis	nconjugatedEstriol	Hemoglobin
Sweat Chloride	Urine Culture	inhibin_A	Platelet Count
IRT	Rubella	Glucose	Gonorrhea
Stool Trypsin	IRT	GTT	Chlamydia
Pap Smear	Stool Trypsin	HIV Antibody	Syphilis
	Pap Smear	Urinalysis	fFN
	HIV Antibody	Hemoglobin	
	Hepatitis B		
	Hemoglobin		
	Sickle Cell		
	CF-Gene Mutation		
	Sweat Chloride		

SADL, as described before, is used to deploy medical sensors, networking sensors and other types of sensors and/or actuators by the monitoring resources providers. These deployed sensors are used by the EHMS to collect data from the pregnant women regarding their medical situation. Figure 9.33 illustrates an example of deploying Haemoglobin sensor with SADL framework, while Figure 9.34 presents the screen shot for the available sensor with sensor and actuator framework.

```
<SADL>
    <Sensor>
        <SensorID>2</SensorID>
        <SensorName> Haemoglobin</SensorName>
        <SensorDescription> Haemoglobin</SensorDescription>
        <ApplicationInformation>
            <App_ID>3</App_ID>
            <App_Name>E-Health</App_Name>
            <Platform>Windows</Platform>
            <Middleware>.Net</Middleware>
            <MaximumUsers/>
            <CurrentUsers>10</CurrentUsers>
    </ApplicationInformation>
    <SesnsorInformation>
            <Type>Blood-Test</Type>
            <DataType>String</DataType>
            <DataStorageType>String</DataStorageType>
            <Host>http://cmpwomar/sensors/Hemo.exe</Host>
            <Container>cmpwomar</Container>
            <Execution/>
            <Status>OnLine</Status>
            <Category>Pregnancy</Category>
            <Location>Liverpool</Location>
    </SesnsorInformation>
    <Contract>
            <ContractID>2</ContractID>
            <CntractName>cmpwail</CntractName>
            <ContractLease>10/10/06</ContractLease>
    </Contract>
    <Interface>
            <InterfaceName>HaemoTest</InterfaceName>
            <InterfaceLocation>http://192.168.1.192/haemoInterf.html
            </InterfaceLocation>
        </Interface>
        <Resources>
            <Processor>PI 233</Processor>
            <Memory>16</Memory>
            <Framework />
        </Resources>
    </Sensor>
</SADL>
```

Figure 9.33: SADL Example for EHMS

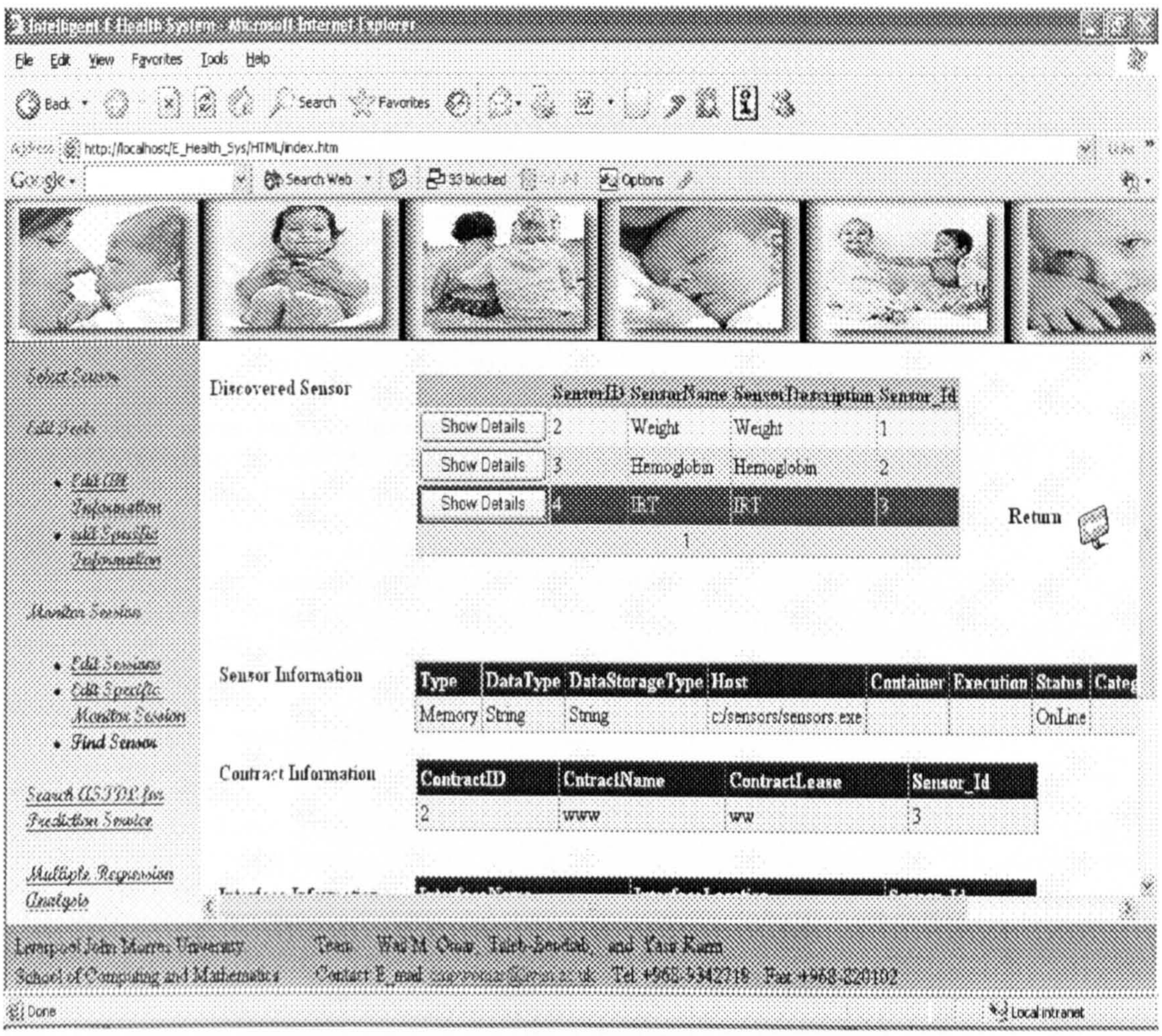


Figure 9.34: The available Sensors In The Sensor And Actuator Container

Hospital employs MSDL for liaise with sensor and actuator framework for discovering the most satisfactory sensors for each pregnancy case. An example of employing MSDL for requesting medical resources are illustrated in Figure 9.35. Figure 9.36 demonstrates the screen shoot of generation MSDL message by the hospital. Sensor schedule service starts looking for the required sensors depending on the obtainable information by SADL and MSDL. Reliability and fidelity of sensor in addition to the validation of the contract are the criterion parameters in searching for the required sensors. Figure 9.37 illustrates the screen shoot for the process of discovering sensors by the sensor schedule service.

```
<MSDL>
  <MonitorSession Session_ID="1">
    <Application>E_Health</Application>
    <Duration>
      <DurationStart>12/28/2005</DurationStart>
      <DurationEnd>21/1/2006</DurationEnd>
      <Interval>2</Interval>
    </Duration>
    <TargetInformaion>
      <Patient_ID>1</Patient_ID>
      <HostName>SU_Wael</HostName>
      <WebAddress>192.168.1.248</WebAddress>
      <UserName>Wael</UserName>
      <Password>Aya</Password>
    </TargetInformaion>
    <ContractInfromation>
      <ContractID>18</ContractID>
      <ContractName>cmpwomar</ContractName>
      <LeaseTime>3/3/2005</LeaseTime>
    </ContractInfromation>
    <Sensors>
      <ID>2</ID>
      <Name>Haemoglobin</Name>
    </Sensors>
    <Sensors>
      <ID>4</ID>
      <Name>IRT</Name>
    </Sensors>
  </MonitorSession>
</MSDL>
```

Figure 9.35: MSDL Request for EHMS

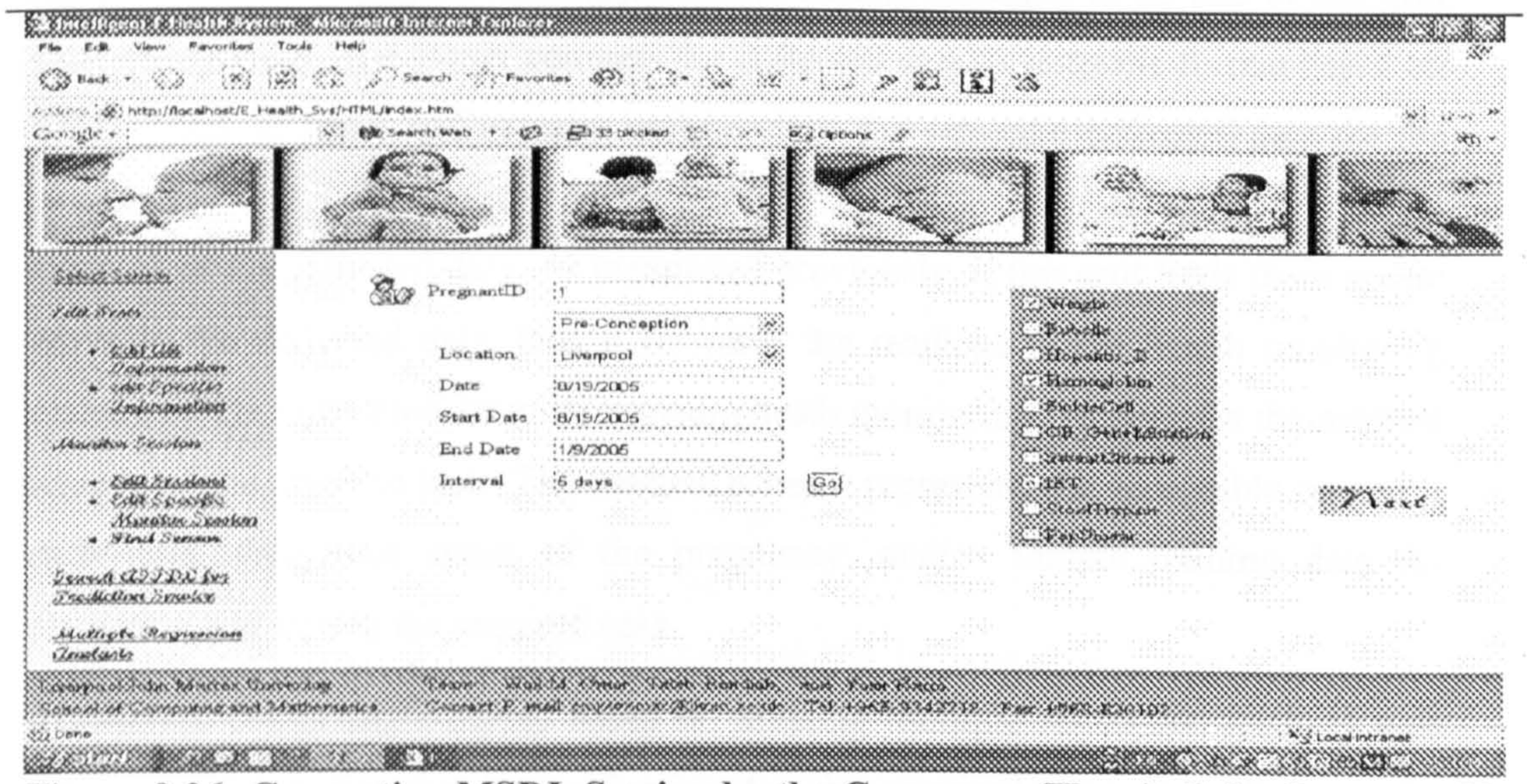


Figure 9.36: Generating MSDL Session by the Consumer (Hospital) for Pregnant ID (1)

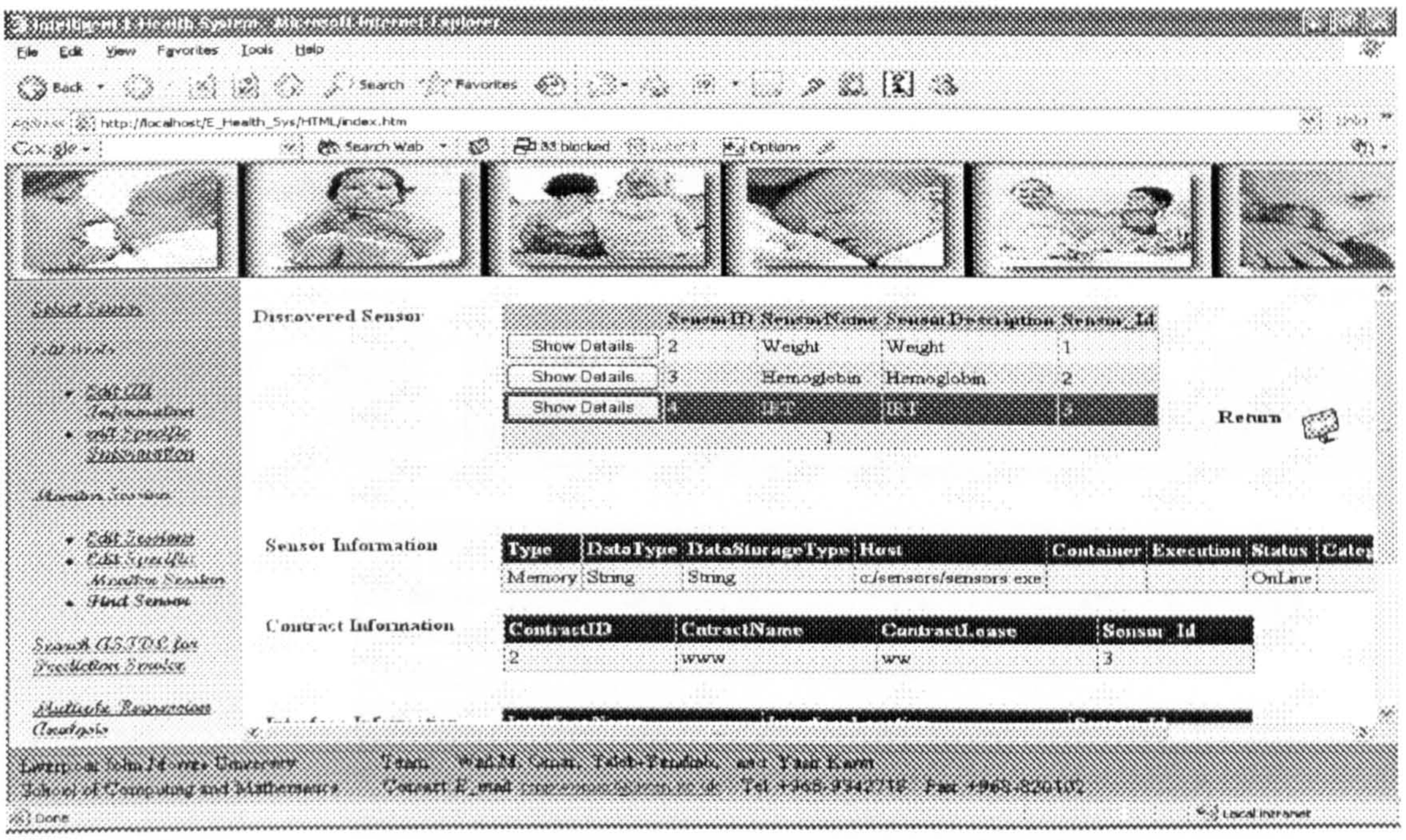


Figure 9.37: Discovered Resources for EHMS

After that, sensor schedule service injects discovered sensors at the home of the pregnant woman which represents the target in this case. Smart homes are proposed to be employed in this scenario to offer a fabric for injecting sensors at the target home. Subsequently, sensors starts collect information and store them in the patient log file inside the logger. The logger in this case is assumed to be one of the grid database services. The logger converts the collected data to semantic format based on XML. Then, analyser and actuator unit receives the semantic collected data for the purpose of checking the validity of the collected data without touching to the medical meaning of the collected data. As mentioned previously, if this unit finds there are no errors in the collected data, then it forwards the readings to the health monitoring system. At the same time, hospital provides heath monitoring system with the medical schema of the required tests. The medical schema represents the acceptable range for each tests, dangerous status of the pregnancy, and/or sample training data for predicting and testing the required case.

Autonomic computing services based on utilising multiple regression analysis web service starts predict the personification for the required case depending on the provided medical schema. The personification status is expressed as good, critical or

dangerous case. This personification is conveyed to the hospital and pregnant women via message alert system.

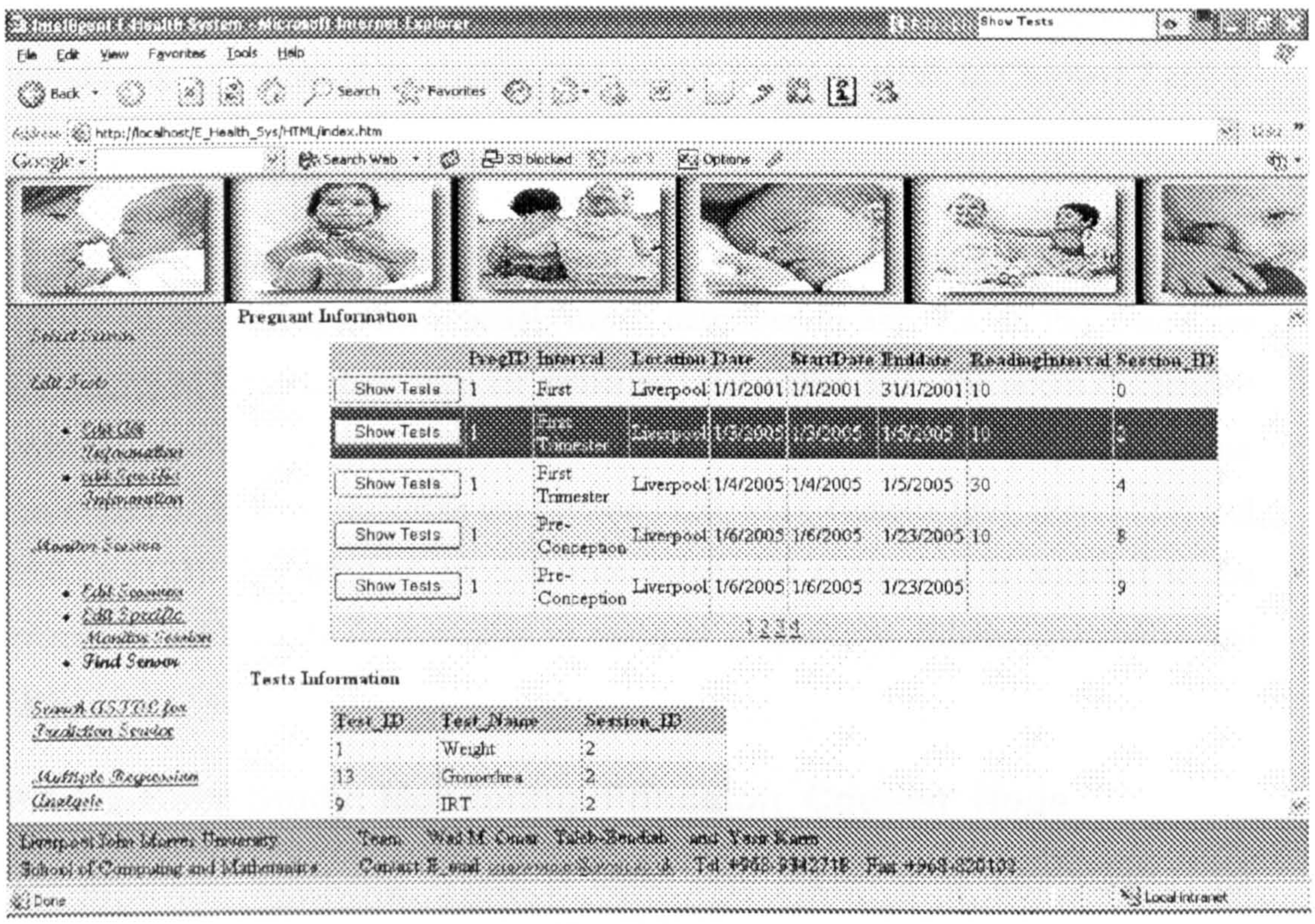


Figure 9.38: List of the Requested Tasks by the Hospital

9.4.3. Monitoring PlanetLab Environment

Situated autonomic computing requires systems to possess and/or be able to access feedback and context information, from their environment, including instrumentation and sensor data. In this section, we are focusing on one framework that is required to support the autonomic services which works inside the agent of the cloud. We are presenting the motivation and development details of a SAF, together with its associated description languages, developed for widely decentralized software systems. To portray our proposed approach, this application example uses an illustrative example taken from an experimental case-study developed using the PlanetLab overlay. The latter is an open community research testbed and overlay for Planetary-scale services.

PlanetLab is adopted as an experimental environment for our developed tools and services. PlanetLab overlay from sensors and services are utilised in order to offer a fabric for supporting information to the autonomic computing services. An introduction of PlanetLab technology and its resources was given in Chapters 2 and 4.

9.4.3.1. System Model

System model for monitoring PlanetLab environment is depending on utilising the developed monitoring system model, which described in Sec. 7.4. In this model we are employed PlanetLab overlay for gathering information from the nodes (targets) in order to sensing the load. Different sensors and actuators are deployed with our developed SAF. Such resources are CoMon [120, 121], Ganglia [39], iPerf [122] and IrisLog [123]. A prediction service from autonomic computing is attached to the system to enhance its functionality in sense of availability, fidelity, reliability and QoS.

9.4.3.2. Case Study: Monitoring 'Princeton_Codeen' Node

The developed monitoring system is tested over PlanetLab to take readings for different parameters from variety nodes. 'princeton_codeen' node with IP address '128.112.139.71' is selected to be the target for our experimental on PlanetLab overlay. In this case study, SADL is used to deploy CoMon sensor as shown in Figure 9.39. The user interface for generating SADL is demonstrated in Figure 9.40.

```
<SADL>
   <Sensor>
      <SensorID>10</SensorID>
      <SensorName>CoMon</SensorName>
      <SensorDescription>To get information from the PlanetLab regarding
      processor, memory, and bandwidth</SensorDescription>
      <SesnsorInformation>
            <Type>Performance</Type>
            <DataType>string</DataType>
            <DataStorageType>XML</DataStorageType>
            <Host>cambridge.intel-research.net</Host>
            <Container>Cambridge</Container>
            <Execution>OnDemand</Execution>
            <Status>On-Line</Status>
            <Category>Memory</Category>
            <Location>planetlab1.cambridge.intel-research.net</Location>
   </SesnsorInformation>
   <Contract>
            <ContractID>16</ContractID>
```

```
        <CntractName>cmpwomar</CntractName>
        <ContractLease>1/1/2006</ContractLease>
    </Contract>
    <Interface>
        <InterfaceName>PlanetLab Sensor</InterfaceName>
        <InterfaceLocation>www.cmpwomar.livjm.ac.uk/
PlanetLabSensor.exe</InterfaceLocation>
    </Interface>
    <Resources>
        <Processor>PI233</Processor>
        <Memory>32M</Memory>
        <Framework>Windows-Unix</Framework>
    </Resources>
    <Application>
        <ApplicationID>12</ ApplicationID>
        <ApplicationName>PlanetLab</ApplicationName>
        <ApplicationMiddleware>.Net</ApplicationMiddleware>
        <Platform>Any</Platform>
        <Middleware>.Net</Middleware>
        <NoOfUsers>50</NoOfUsers>
        <CurrentUsers >10</CurrentUsers >
    </Application>
  </Sensor>
</SADL>
```

Figure 9.39: SADL Example for PlanetLab Environment

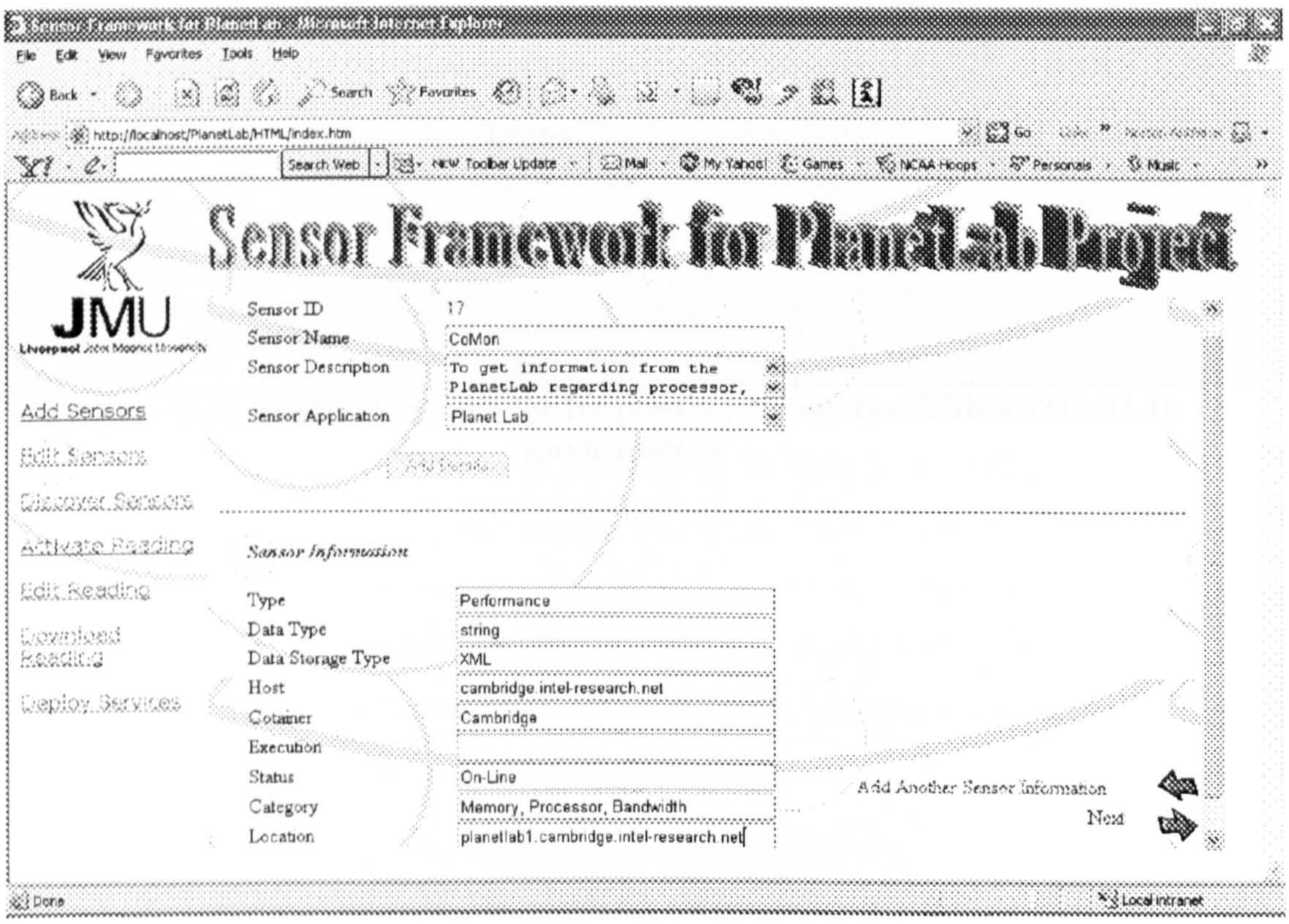


Figure 9.40: User Interface For Generating SADL in PlanetLab Environment

After deploying the monitoring resources with the framework, consumer utilises MSDL to generate and send the request of collecting information from the target 'princeton_codeen' to the SAF, as illustrated in Figure 9.41. The consumer in this case should select PlanetLab as an environment application. The durations tag is set to 10 minutes and 20 seconds for each trigger of collecting data. The user interface for creating MSDL is shown in Figure 9.42.

```
<MSDL>
 <MonitorSession Session_ID="10">
 <Application>PlanetLab</Application>
 <Duration>
      <DurationStart>11:05:49 AM</DurationStart>
      <DurationEnd>11:07:49 AM</DurationEnd>
      <Interval>20000</Interval>
 </Duration>
 <TargetInformaion>
      <Client_ID>544</Client_ID>
      <HostName> princeton_codeen </HostName>
      <WebAddress>128.112.139.71</WebAddress>
      <UserName>Wail</UserName>
      <Password>XXXXXX</Password>
 </TargetInformaion>
 <ContractInformation>
      <ContractID>16</ContractID>
      <ContractName>cmpwomar</ContractName>
      <LeaseTime>1/1/2006</LeaseTime>
 </ContractInformation>
 <Sensors>
      <ID>10</ID>
      <Name>CoMon</Name>
 </Sensors>
 </MonitorSession>
</MSDL>
```

Figure 9.41: MSDL Example for Requesting Resources Within PlanetLab Environment

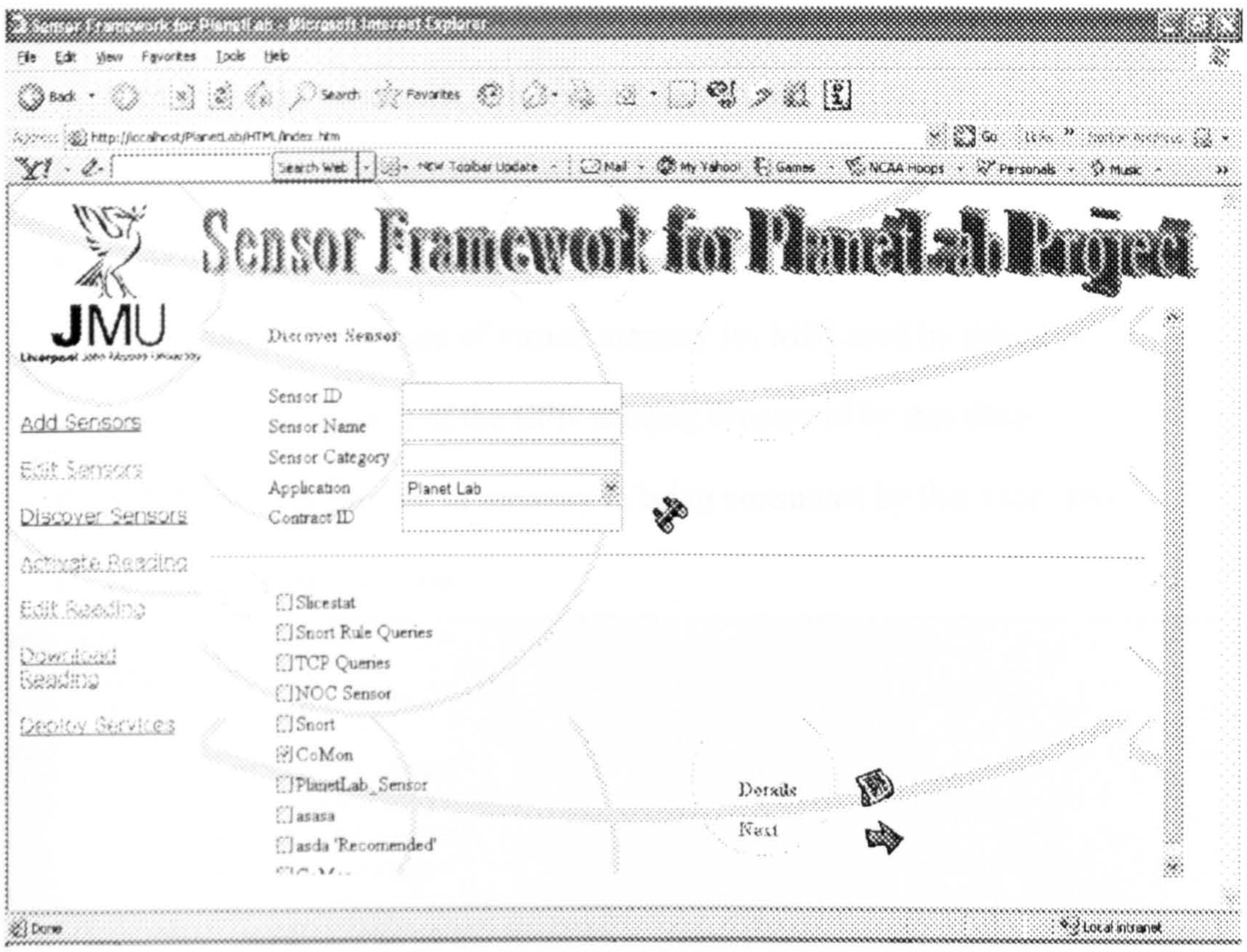


Figure 9.42: User interface for Generating MSDL for PlanetLab Environment

The SAF analyses the MSDL request and starts search for the required sensors. Autonomic computing is employed in the case to find the best match for consumer's request. CoMon in the framework container is selected by the system as the best choice for our demands. The system injects CoMon in the target. CoMon begins gathering information on-fly and on run time from 'princeton_codeen' node. The collected data is saved then in the Logger. The logger converts the reading to XML format to be acceptable in heterogeneous system, as shown in Figure 9.43. The user interface for editing information in logger is shown in Figure 9.44. The consumer has the facility to download the readings as XML format. The data now is ready to be delivered to the consumers. The collected parameters include:

- **CTX:** the slice's context ID, a sort of user number
- **TX1:** transmit bandwidth in Kb/s over 1 minute
- **TX15:** transmit bandwidth in Kb/s over 15 minutes

- **RX1:** receive bandwidth in Kb/s over 1 minute
- **RX15:** receive bandwidth in Kb/s over 15 minutes
- **#PR:** the number of processes owned by this slice
- **PMEMMB:** the amount of physical memory (in MB) used by this slice
- **VMEMMB:** the amount of virtual memory (in MB) used by this slice
- **%CPU:** what fraction of the CPU is being consumed by this slice
- **%MEM:** what fraction of memory is being consumed by this slice, and
- **NAME:** the slice's name.

```
<Logger>
   <MonitorSession>
      <Mon_ID>1</Mon_ID>
      <Readings>
            <CTX>524</CTX>
            <TX1>66</TX1>
            <TX15>151</TX15>
            <RX1>104</RX1>
            <RX15>176</RX15>
            <PR>58</PR>
            <PMEMMB>79.2</PMEMMB>
            <VMEMMB>147.0</VMEMMB>
            <CPU>18.8</CPU>
            <MEM>7.8</MEM>
            <Name>princeton_codeen</Name>
            <TimeReading>2/17/2005 12:49:55 PM</TimeReading>
      </Readings>
      <Readings>
            <CTX>524</CTX>
            <TX1>66</TX1>
            <TX15>151</TX15>
            <RX1>104</RX1>
            <RX15>176</RX15>
            <PR>58</PR>
            <PMEMMB>79.2</PMEMMB>
            <VMEMMB>147.0</VMEMMB>
            <CPU>13.2</CPU>
            <MEM>7.8</MEM>
            <Name>princeton_codeen</Name>
            <TimeReading>2/17/2005 12:50:12 PM</TimeReading>
      </Readings>
   </MonitorSession>
</Logger>
```

Figure 9.43: Logger Example for PlanetLab Environment

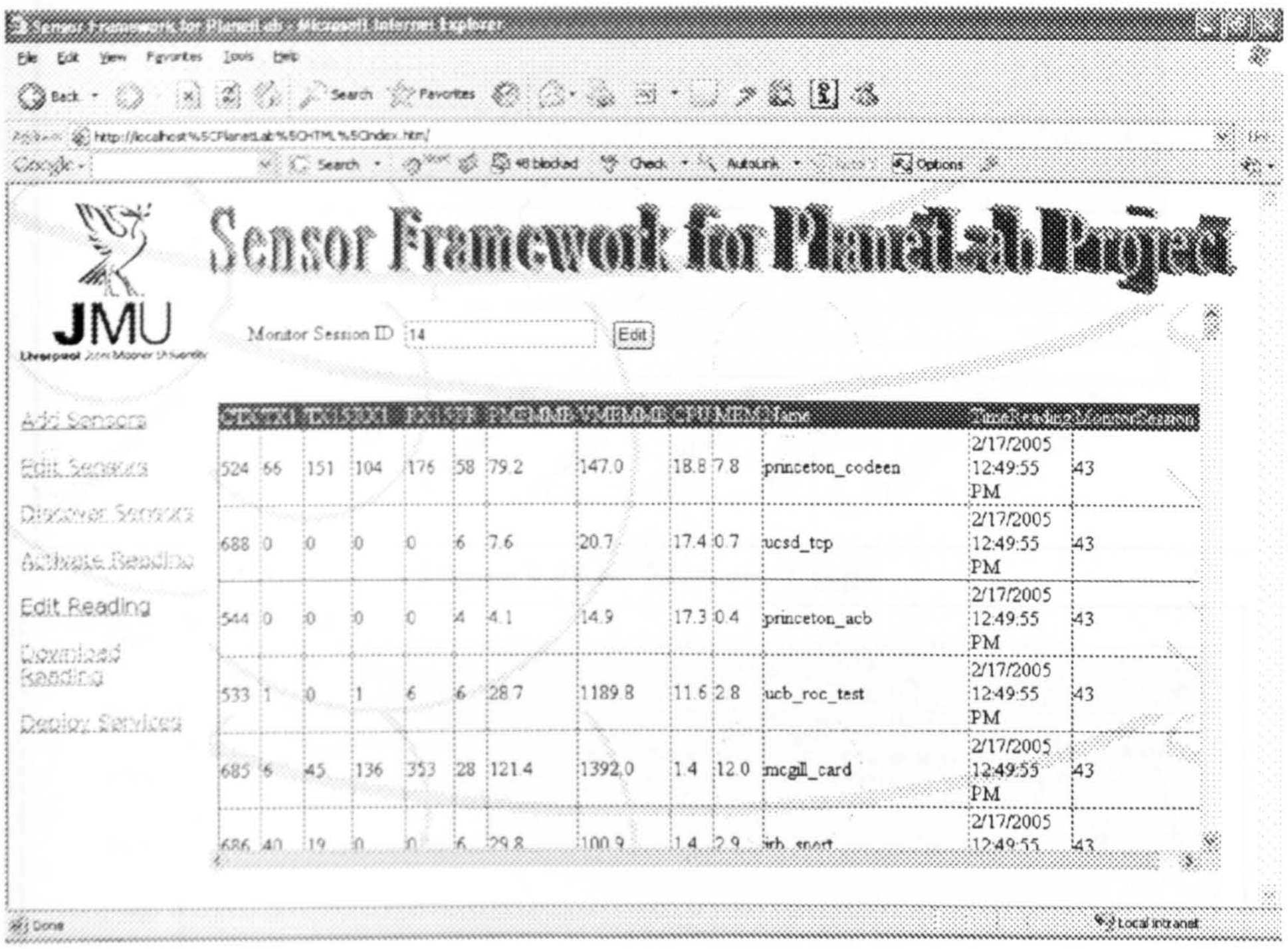


Figure 9.44: Collected Data Inside the Logger for PlanetLab Environment

Figures 9.45.a, b, c and d demonstrate the CPU, memory, virtual memory and bandwidth usages respectively for reading time equal to 10 minutes with interval 20 seconds between reading and the other for node 'princeton_codeen', as mentioned-above.

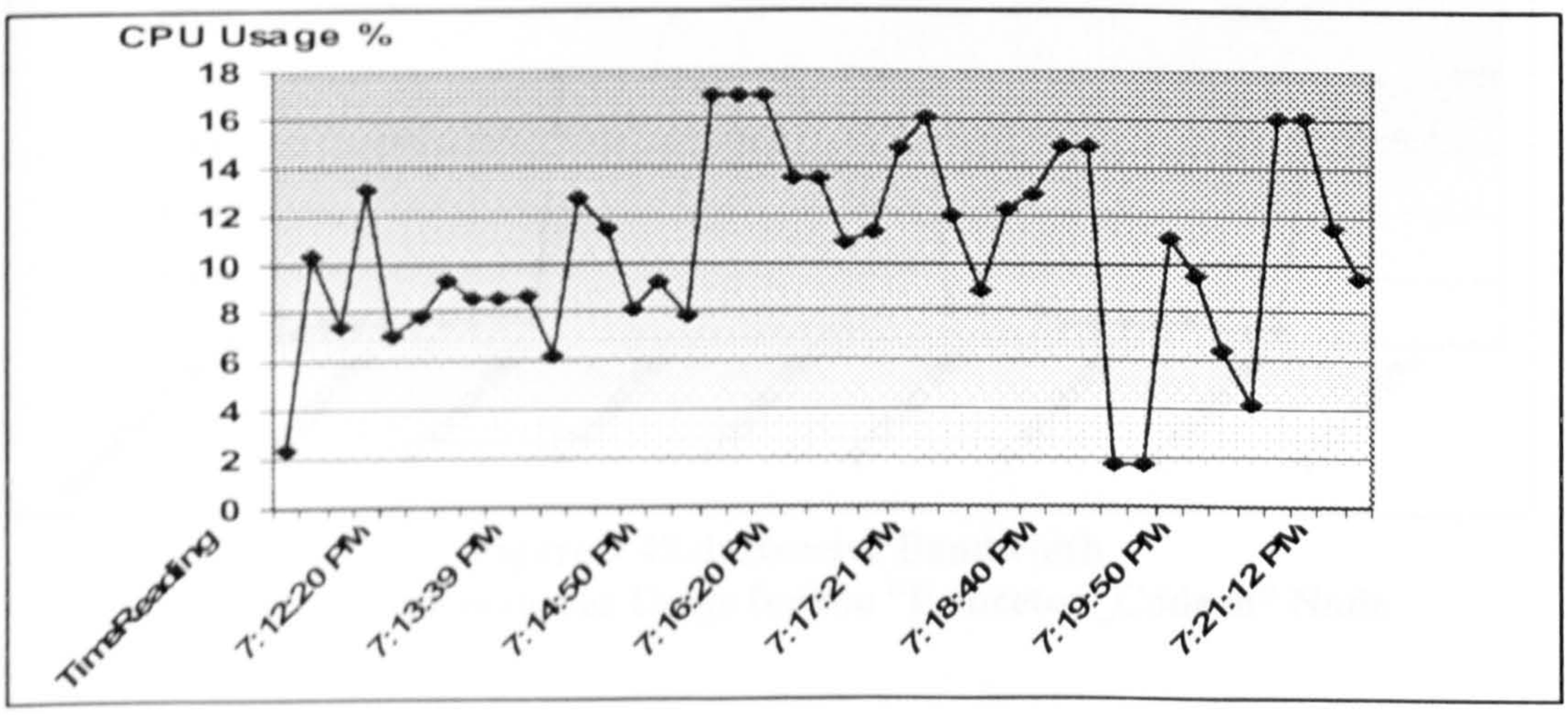


Figure 9.45.a: CPU Usage

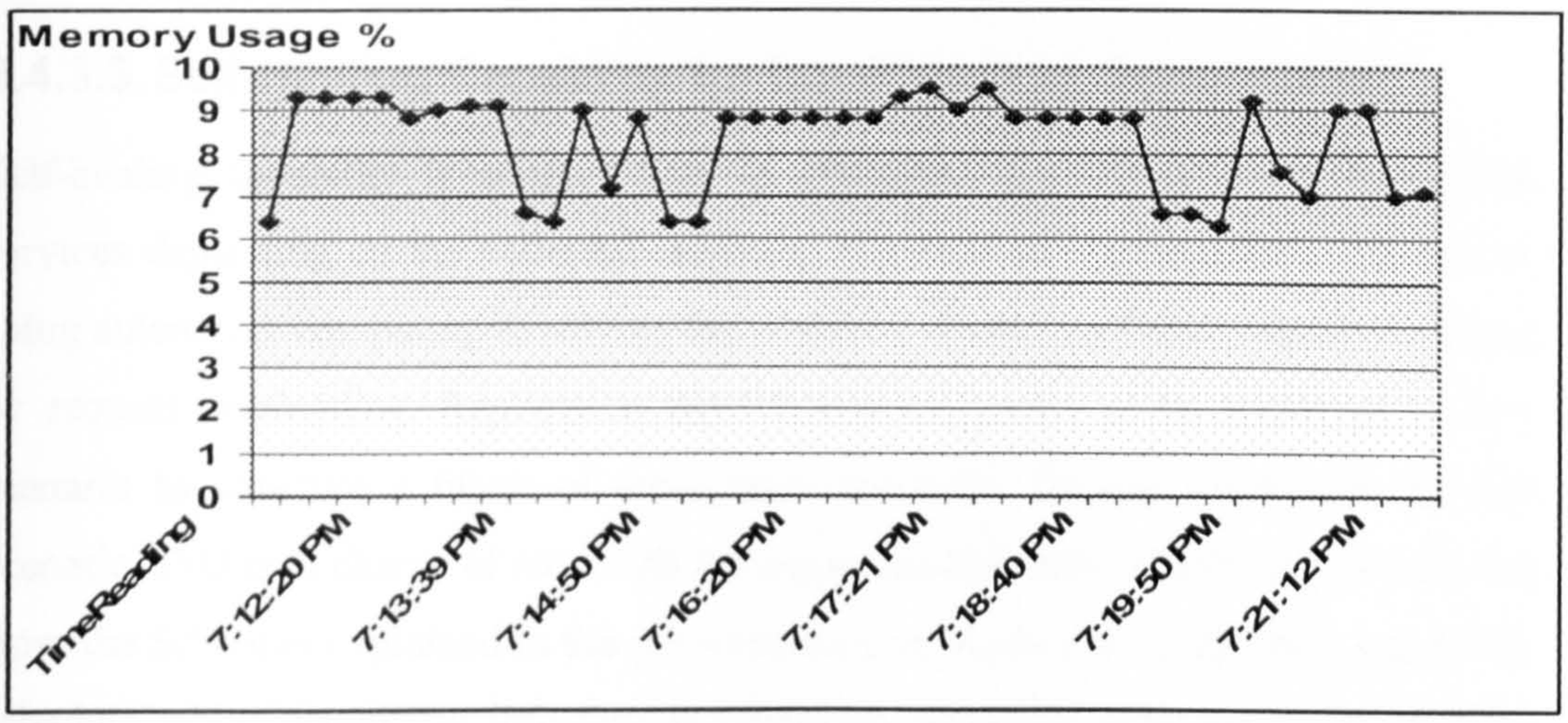


Figure 9.45.b: Memory Usage

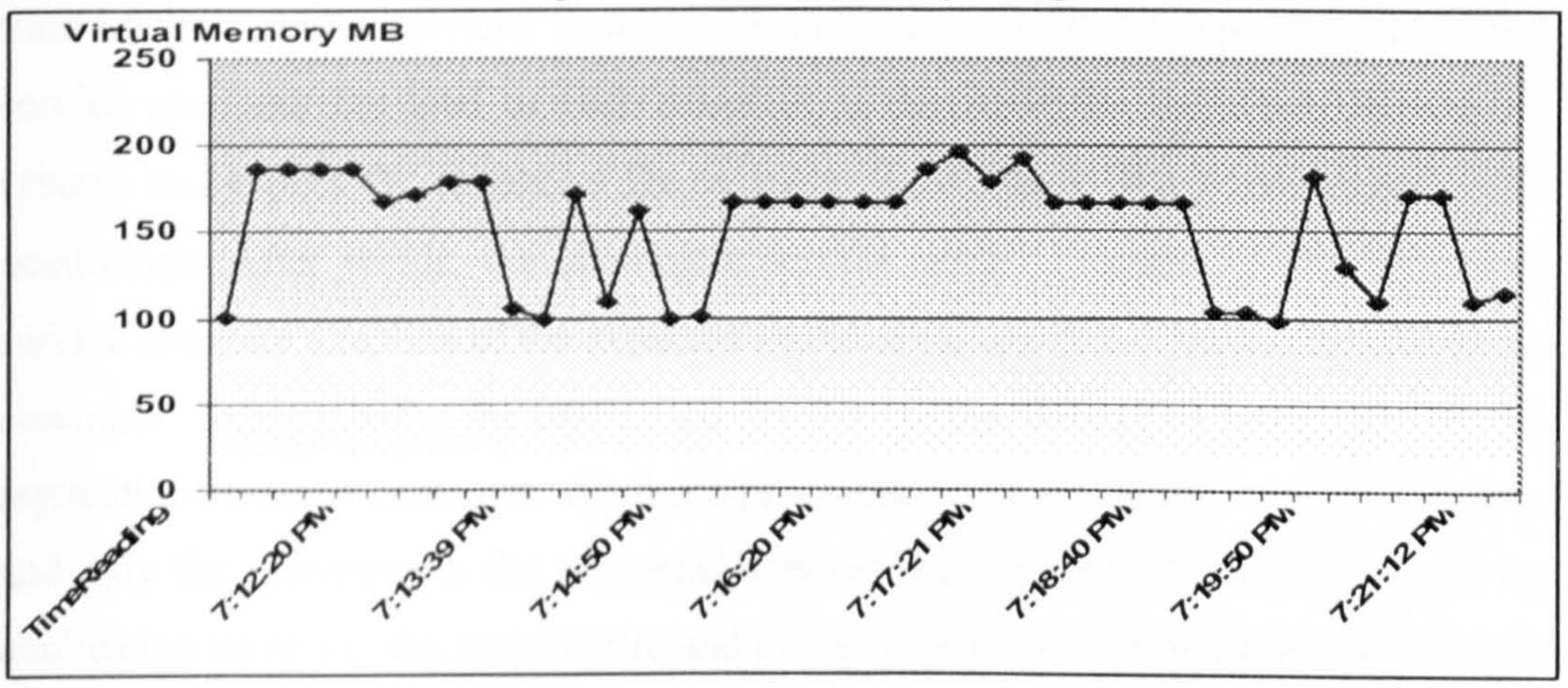


Figure 9.45.c: Virtual Memory Usage

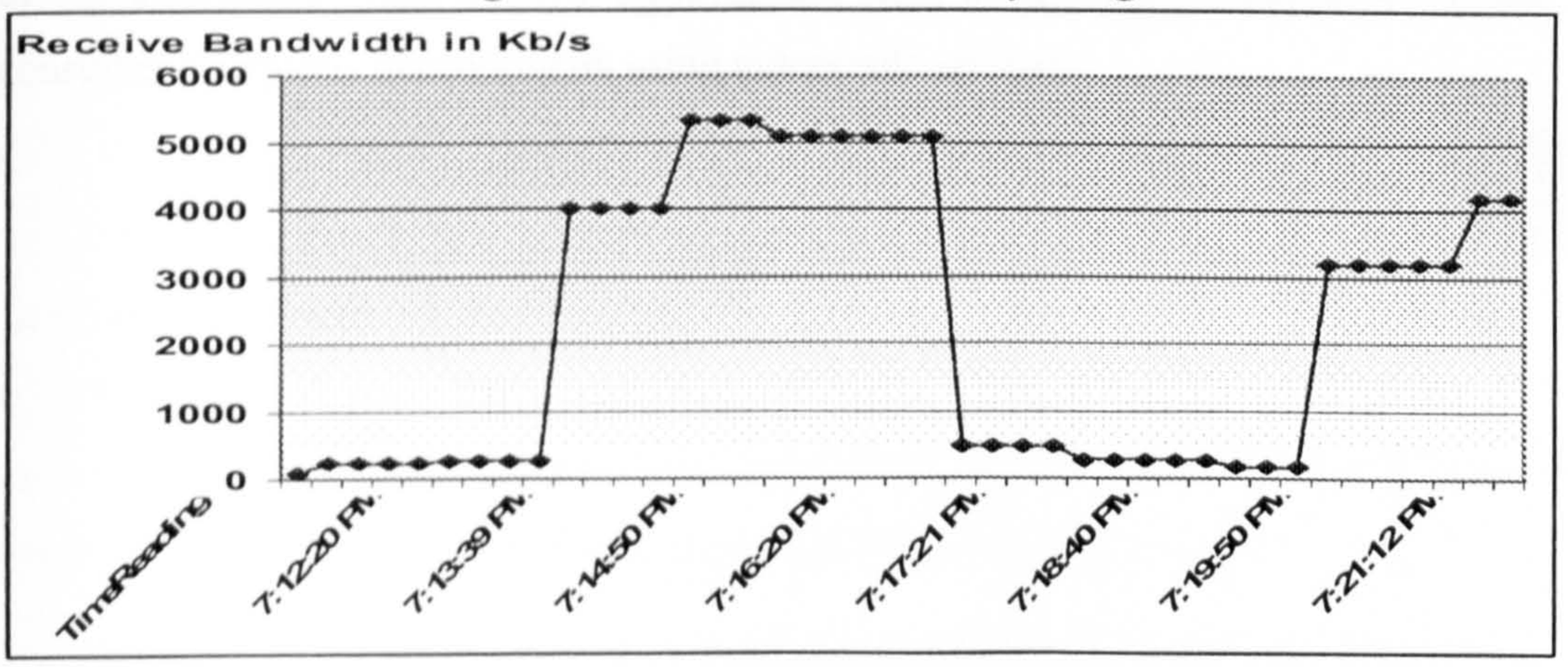


Figure 9.45.d: Receive Bandwidth

Figure 9.45: Resources Usage for the "Princeton_Codeen" Node

9.4.3.3. Self-Healing Capability for the PlanetLab Environment

Self-healing capability is adopted here to overcome the failure in the PlanetLab services depending on the collected readings. Figure 9.46 illustrates the scenario of using autonomic computing in solving the problem of services failure which is caused by request overloading. Replication algorithm is suggested to be employed in this scenario to structure a fabric of emergency resources for the consumers. In this scenario, JSU is in charge of record all the resources demands, which can be obtained from the SAF that explained in the previous section. Each one of the JSUs sends the schedule tables containing information regarding requested resources and schedule time of the running services to the autonomic services. In its turn, the autonomic service estimates the load of each resource in the resources container. Hence, the system can expect the failure of the resource according to the overload before its occurrence. After words, the autonomic service sends a request to the replication service to create a replica of the expected resource failure in the *replication resources container*. Subsequently, the replication service creates demanded resources into the replication resource container. All these processes occur inside the middleware layer and only the responses to the requested services are passed to the consumers in the application layer, i.e. the middleware and its services work as a black box in the sight of the consumers. This method achieves the high availability of resources to the consumers' requests depending on using autonomic services.

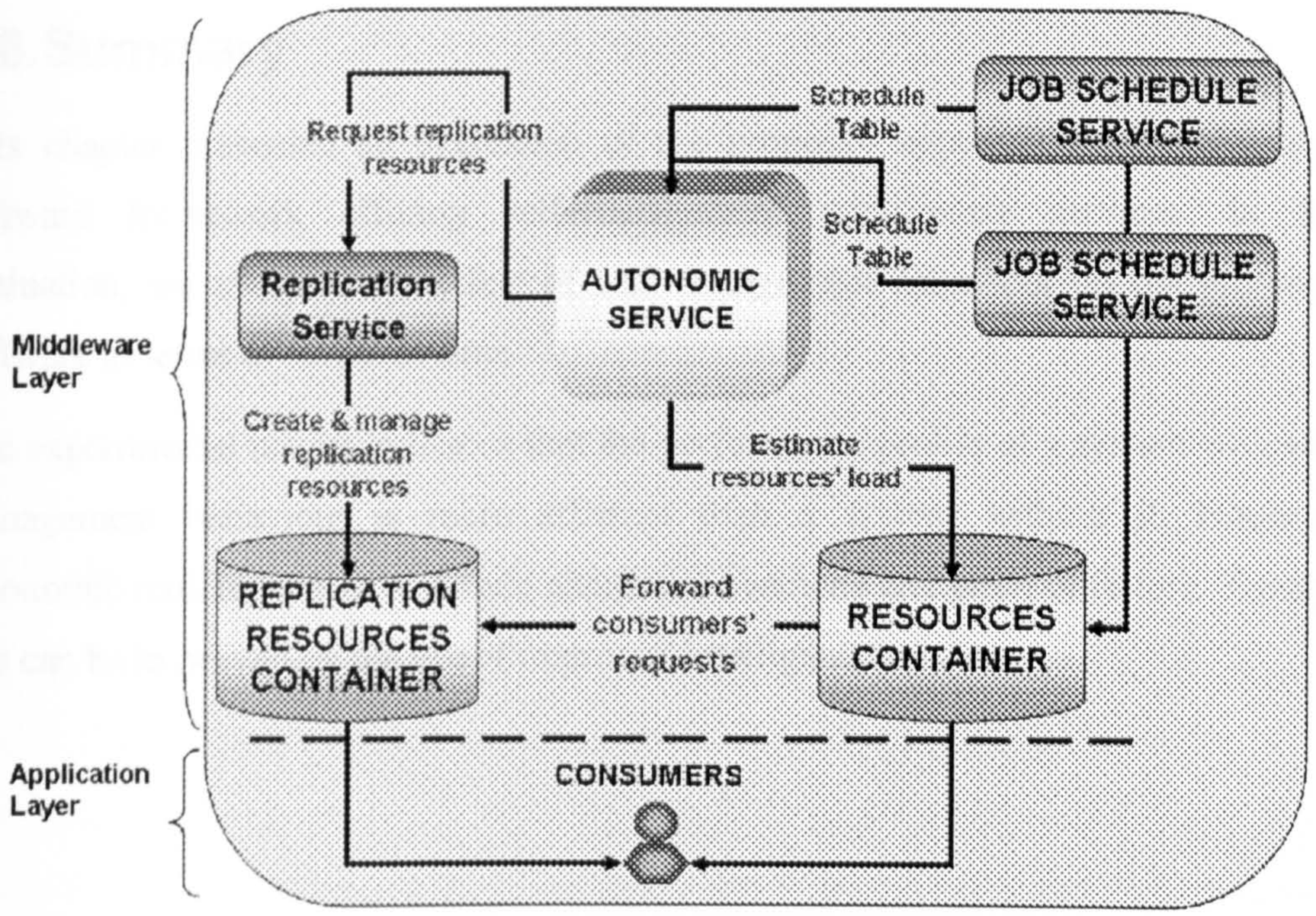


Figure 9.46: Life Cycle for Self-Healing System

9.5. Discussion

It is a significant challenge to produce conclusive evidence of the benefits, merits, effectiveness, correctness and completeness of the proposed models, framework, services and utilities. However, both the qualitative and quantitative evaluations provide positive indications that the proposed models and associated programming model seem to fulfil the defined requirements, and are generic and flexible enough to support the rapid development of a range of distributed applications. In particular, in this evaluation, based on a number of experimental actions, the assembly services succeeded in deploying, discovering and invoking distributed resources (services, infrastructures, sensors and actuators). Moreover, the embedding of autonomic computing services in enterprise applications improves the performance of the self-management system in sense of fidelity, reliability, availability and QoS.

The potential benefits of autonomic computing vision to support planetary scale enterprise applications have been indicated through a set of examples, namely; (i) the On-Demand Services (ODS), (ii) Remotely Patient Monitoring in E-Health Monitoring System (EHMS), (iii) Monitoring Testbed for PlanetLab Environment.

9.6.Summary

This chapter presented an evaluation of the proposed approach for developing a software framework offering self-management middleware services. In this evaluation, we analysed the effect of self-management behaviour on the prototype software in terms of response time.

The experimental results indicated that the performance profile of a system with self-management behaviour is more efficient than a system without it. However, autonomic middleware services add additional overhead and point of failure. Though, this can be improved by the use of improved intelligence services.

CHAPTER 10

CONCLUSIONS

10.1.Motivations and Approach

Recent advancements in networking, hardware, and middleware technologies have been a major catalyst for the recent popularity of grid-based applications [33], which are typically characterized by their high-performance computing requirements and dynamic resources transparency. To achieve this vision, a huge numbers of services and infrastructures are required to be invested in such global computing environment. Therefore, many are anticipating grid computing infrastructure, utilities and services to grow dramatically in size and functionality and become an integral part of future socio-economical fabric. Though, this vision is predicated on that such grid-based computing services and infrastructure have to ensure a high-assurance, dependability guaranties, interpretability, and ubiquitously of resources, whilst lowering complexity, cost and eases of use.

From the above addressed points, we can conclude the difficulties that can be faced in developing global computing environment. Such difficulties are heterogeneous environment, the absence of the centralized system, dynamic infrastructures, sharing of resources, security policies, management rules and strategies, variety of network/connection protocols and absence of a common data representation. The suggested solution for much of these difficulties is the self-management system that is integrated with planetary-scale system and its applications in order to generate viable system which is able to survive within the changing in the behaviour and actions of the environment taking in the consideration the boundaries of the applications' nature.

Such essential requirements for self-management system have brought the attention to the autonomic computing capabilities as a vital technology to underpin such vision of

viable flexible global computing system. Autonomic computing as other similar initiatives before it [88], advocates the delegation of much of systems adaptation, management, tuning, and protection to the software itself.

Much of this work investigates the designing, developing and implementing of the autonomic computing capabilities as one of the middleware core functions. Moreover, this effort covers the generic models and design and implementation requirements for the usability of autonomic computing services in planetary-scale system. The merge of autonomic computing and global computing technologies is proposed in order to reduce and hide the complexity of the environment from the consumers by reducing the interaction and move to the automated epoch in carrying out the tasks. Many tools, services and frameworks are addressed through this research in order to assist the autonomic computing (intelligent part of the system) to establish self-managing system that can be embedded with the environment and/or applications.

The creation of this kind of system involves a range of technical issues need to be addressed and developed, which cover:

- **Reference Models:** the development of generic design models and architectures as well as patterns to support software developers and middleware services to develop systems, which have the feature of self-awareness (monitors) and runtime self-management facilities. This is based on a developed method for automated service assembly and deployment.
- **Experimental Insight:** to demonstrate the developed middleware services which can perform number of core functions such as deploy, discovery, invocation and management.
- **Open Standard Format:** to offer formalise and semantic way in exchanging information between different actors of the system. Such open standards provide a common language between the consumers, resources providers and frameworks.

In line with the above described motivations and associated challenges, this thesis detailed a proposed software framework that offering a number of middleware services for: (i) Self-management system (ii) Self-monitoring and diagnostic system.

For theoretical support, the research visited a number of fields including:

- **Autonomic computing model and approaches:** using autonomic computing capabilities to design and develop the policies and strategies which required for establishing survival self-management system for planetary-scale system.
- **Advanced software engineering:** using distributed middleware as a broker to facilitate the communication and the coordination between both base services (users application service) and the meta-services (autonomic middleware control services), and to bridge the gap between network layer and the transport and application layers (from the Internet model) according to the developed network model for the global computing.
- **Service-oriented developments:** using the concepts of components, connectors, and services to generate a system from number of distributed components. Moreover service-oriented developments viewing the services as discoverable logical entities that are defined by published interfaces.

In particular, this work provided support for global computing vision by merging both practical and theoretical concepts, which can be concluded as:

- **The practical support:** is providing the essential and required services and infrastructure for developing the computational and programmable model for autonomic middleware control service leading to the goal of self-management system.
- **Service management:** is responsible for the life cycle of the services starting from deployment phase and ending with passing results to the consumers.
- **System management:** outlines the plane, policies (rules) and strategies for performing automated and self-management services.
- **Distributed shared space:** allows all system' services and infrastructures to be shared over the network in addition to provide the control and audit system with the required information.

10.2.Achievements and Contributions

This work makes a number of contributions towards a better understanding of software self-management requirements for the global computing environment. One of the main contributions of this work is the generic design model for planetary-scale environment based on existing Internet networking model. This developed model is vital for providing a way for managing planetary-scale system by separating the systems into numbers of clouds/zones managed by an agent that runs the middleware layer. The later is responsible for performing managing processes.

In addition, the work provided an insight into the design of patterns for the generic model of viable self-management system for the planetary-scale environment. This model is designed based on merging the models of two patterns that are accepted in software engineering community, which are Viable System Model (VSM) pattern and Gang of Four (GoF) pattern. Moreover, the Self-Management Viable System Model (SVM-SM) is developed to achieve the general vision of the global computing which is moving to the next generation of self-governance distributed system. In addition to the above contributions, this work had addressed the following point:

- **Consolidate of autonomic computing capabilities:** varieties of autonomic computing capabilities are proposed in this research in order to corporate together for offering a therapy for global computing to perform self-management and self-governance system (Sec 5.5).
- **A mechanism for implementing intelligent web services:** different algorithms, approaches and techniques for the predication and classification processes are surveyed and developed as web service to be the core unit of the distributed autonomic computing within OGSA environment (Sec 7.2).
- **Services and infrastructures framework:** to offer a public container for storing variety types of resources. This framework is developed in the concept of offering a manageably fabric that guarantee the QoS and availability of resources (Sec 7.3).

- **Services and infrastructures description language:** to be a common language between the resources providers and the consumers. Such formalised language fit with the requirement of open standard community a long with providing the tools for supporting the fidelity and assurance of selecting aimed resources (Sec 8.2).

- **Monitoring model:** to offer a model for utilising monitoring system that can be integrated and requested by any components of the auditing system or enterprise applications (Sec 7.4). This model provides a framework for the generation, deployment, discovery, invocation and management of monitoring resources. For this purpose, two description languages are proposed and developed, namely; Sensor and Actuator Description Languages (SADL) (Sec 8.4) and Monitor Session Description Language (MSDL) (Sec 8.5). These are used respectively to describe the set of deployed sensors and actuators in a given self-managing planetary-scale system infrastructure, and to define monitoring properties and policies of a given monitored target application services.

- **Evaluation:** Self-management model is tested and evaluated through a number of quantitative and qualitative evaluation processes in order to provide a proof-of-concept or evidence of the potential benefits of such autonomic middleware control services (meta-control model) and associated baseline architecture to distributed application life-time management and viable self-management system. Three practical enterprise applications are used to show the usability of our model in real world applications. These applications are:

    - *On-demand services* (Sec 9.4.1): is a way to reduce the interaction between the users and the resources by adjusting the distributed resources to give superior services to the users. On-demand service is implemented based on autonomic computing capabilities in order to manage the consumers' usages. Service reservation and schedule service is presented in this application in order to serve the consumers' requests in automated and smart way.

- *E-Health monitoring system* (Sec 9.4.2)*:* to offer a health monitoring tools for auditing patients remotely which assist in providing the hospital (consumer) by online and on-demand follow-up services. E-health monitoring system depending on autonomic computing capabilities are adopted in this work to predict the personification for each case and proposed the required sensors that need to be injected in patients' side.
- *Monitoring PlanetLab environment* (Sec 9.4.3): to possess and/or be able to access feedback and context information that is required by situated autonomic computing service. PlanetLab overlay from sensors and analysers are employed for collecting and analysing distinction information from Virtual Servers, Slices, Nodes and other parts of the PlanetLab environment.

10.3.Thesis Summary

Grids are emerging as the infrastructure for next generation of the Global computing. In such environments, resources are heterogeneous and geographically distributed with varying availability, usage and cost policies. Hence, this thesis has offered a new vision of lifetime management of distributed application services grounded in a number of related disciplines such as; self-management patterns and model, software agent, monitoring system and advanced software engineering. The detailed description of background theories, methods and the achievements of this work are presented as follows;

- Chapter 1 introduced the motivations and technical challenges and outlined the proposed approach and main contributions of the work.
- Chapter 2 introduced the required basic background concepts and principles of planetary-scale system including the basic definition of the grid computing followed by its components, capabilities, architectural model and topologies. Moreover, this chapter presented Open Grid Service Architecture (OGSA) and PlanetLab environments as parts of the global computing environment.

- Chapter 3 described the basic background and concepts of autonomic computing model. This background covered autonomic computing standards and architecture, characteristics and capabilities. Moreover, this chapter introduced the monitoring system which is essential for completing the life cycle of the self-management system.

- Chapter 4 reviewed the state-of-the-art and related work relevant to the control and management aspects for the planetary-scale system. This review covered autonomic grid computing applications and autonomic computing models and approaches. Most of such research works have been focused on self-management middleware system. At the end, this chapter outlined the existing monitoring resources for the planetary-scale system

- Chapter 5 outlined the requirements and models for designing self-management middleware services. Moreover, requirements and model for autonomic computing and the interference between autonomic computing capabilities were demonstrated in this chapter.

- Chapter 6 demonstrated the generic model for the planetary-scale system. Moreover, this chapter presented Self-Management Viable System Model (SM-VSM) for the planetary-scale environment, which was introduced from the modification of the Viable System Model (VSM). SM-VSM consists of 5 layers which are policy, intelligence, control and auditing, coordination and operation. For the purpose of the distributed system, we added another intermediate layer between the operation and coordination layers. This intermediate layer is known as support distributed functions layer. In addition the corporation between SM-VSM and Gang of Four (GoF) patterns were described in this chapter.

- Chapter 7 illustrated the support utilities for designing, developing and implementing autonomic computing services as one of the core middleware services. Intelligent web services for autonomic computing were designed based on utilising of machine learning technique and multiple regression algorithm. This was followed by description of the sensor and actuator

framework. Monitoring model was presented in this chapter, which includes sensor and actuator framework and the required description languages.

- Chapter 8 presented our technique for the open standard based on the use of eXtensible Mark-up description Language (XML). Three types of description languages were described in this chapter. The first description language was Assembly Services and Infrastructures Description Language (ASIDL), which was designed for the purpose of providing common language between consumers and resources providers. On the same concepts two other description languages were produced to offer a formalised language between monitoring resources and monitoring requesters. These two languages were Sensor and Actuator Description Language (SADL) and Monitor Session Description Language (MSDL).

- Chapter 9 presented a qualitative and quantitative evaluation of the main functionalities of the framework services. For the quantitative evaluation we used three enterprise global application examples namely; on-demand service, E-Health monitoring system and monitoring PlanetLab environment. Experimental results showed that the proposed models and techniques are efficient and effective in carrying out self-management processes.

- Chapter 10 provided the thesis motivation and approach, achievements and contributions, summary and suggestions for further work.

10.4.Conclusion and Discussion

Internet model has been used in the current Internet environment to offer a method for describing and analyzing the transfer of messages between end-systems. Between the many already identified shortcomings of this networking model is the absence of reliability, fidelity and high availability of resources. Therefore, this model should be comprehensive in order to include the description of different methods and functions that are required for managing and automating the deployment and discovery processes for distributed resources in order to provide better services and

performance. This work adopts Internet model as a start point for developing a new Network model that can achieve the new needs of global computing.

Middleware layer is proposed in this research to be attached between transport layer and network layer from the Internet model. The middleware layer is developed in order to be responsible for managing the consumers request and resources in a way that assures the interpretability, ubiquitously, and high availability along with specifying security and ownership polices. Middleware layer consists of three sub layers namely; Serviceware, core-functions and resources overlay.

Serviceware sub layer is responsible for performing the self-management processes based on utilising inter functions represented by autonomic computing services. In this research, we designed a pattern model for self-management system in order to assist the system in defining and selecting the basic components that are required to carry out the management task taken in the consideration the nature boundaries of the environment. Moreover, the developed patterns help the system to survive within irregular environment such as grid or other type of planetary-scale system. Viable System Model is employed in this case after a little modification to be adequate for the large scale enterprise applications and environment. This model describes the process of generating self-management pattern till it reaches to the basic operations. These operations are represented by autonomic computing capabilities. Therefore, other patterns are required to describe the fundamental components and operations of the autonomic computing capabilities. For this purpose, Gang of Four (GoF) patterns are employed to define the design of these basic components.

The other part of the story is that autonomic computing capabilities which require a number of services, tools and frameworks in order to perform the task of self-management. Such utilities are required to collect information from the environment and perform actions inside the environment. Therefore, monitoring model is developed, designed and implemented in order to deploy, discover, invoke and manage the monitoring resources from sensors and actuators for the purpose of collecting wealthy data for the autonomic computing services. This monitoring model is extended to be suited for embedded with any enterprise applications.

Open standards concept is one of the most support concepts for the future global computing applications. Therefore, three description languages are designed to provide a fabric for formal exchanging of information between actors of the system. These description languages, as mentioned before, are Assembly Services and Infrastructures Description Language (ASIDL), Sensor and Actuator Description Language (SADL) and Monitor Session Description Language (MSDL).

In this research, we applied our development models and approaches on three enterprise applications in order to be sure that these developing materials are feasible for managing the future global computing environment and structure. These enterprise applications are on-demand service, E-Health monitoring system and monitoring PlanetLab environment. The developed models, services, tools and frameworks are embedded with these systems without any major changes. This gives an indication of the feasibility of requesting and using these elements with most of the enterprise applications.

10.5.Proposed Further Works

This work strived to be comprehensive and novel in providing a model for self-managing global computing and its applications. However, this work can not cover all approaches and techniques that can be used in performing the required tasks or the enterprise applications that can benefit from this model. Therefore, the following points outline suggested further works including;

- The developed models, services, tools, frameworks and description languages have been tested and evaluated through three enterprise applications. These can be extended to include more applications and models, such as:
    - ASIF and ASIDL have been used to deploy, discover, invoke and manage the resources for ODS and EHMS. This framework with its associated description language can be used and experimented for deploying different types of resources without depending on the use of UDDI. The intelligence can be annexed to this framework in order to enhance it ability for interacting with environment.

- The use of the ODS technique for overcoming the long processing time for some of the large enterprise applications.
- SAF and its description languages (SADL and MSDL) have been developed for auditing the behaviour of the three selected applications in this work. These modules can be extended to monitor other type of environment and applications in a way that would provide a wealth of information for the administrative services. Such environments and applications are GridBus [156], distributed educational software, e-business environment, and bioinformatics applications.
- The use of the developed intelligent services (SOM and Multiple Regression) as a predictor for variety of autonomic computing capabilities.

- **Design Patterns:** to investigate the feasibility of integrating others levels of GoF patterns with SM-VSM in order to describe, develop and implement the basic components of the autonomic computing capabilities. These levels of GoF patterns are represented by behavioural and structural patterns.
- **Environment Modelling and Monitoring:** to investigate environment modelling, embodiment, scanning including policy-based management and monitoring. Also, exploring the opportunity to use the developed monitoring model with its framework, description languages and dynamic resources with variety types of sensors for different large-scale enterprise applications.
- **Autonomic Computing Capabilities:** to investigate other types of autonomic computing capabilities to extend the proposed autonomic middleware services, utilities and frameworks. This can include studies of intelligent services like support vector machine [131, 157], Neural Network [158, 159], Bayesian network algorithm [160, 161].
- **Sensor and Actuator Description Language:** to investigate the integration of this description language with the Web Services Distributed Management (WSDM) [162] in order to support the process of managing web services.

APPENDIX A

MIDDLEWARE

To achieve the true benefits of distributed systems approach including; peer-to-peer and client-server, developers must have a set of tools that provide a uniform means and styles of access to system resources across all platforms (heterogeneous system). This will enable programmers to build applications that are not only look and feel the same on various PCs and workstation, but that use the same method to access data and resources regardless of the location of data and resources.

The most common way to meet this requirement is by the use of standard programming interface and protocols that lie between the application above and communication software and operating system below. Such standardised interfaces and protocol have come at top and be referred as *middleware*.

There are a variety of middleware solutions ranging from the very simple to the very complex. All types of middleware share the same concept of hiding the complexity and disparities of different network protocols and operation systems from the consumers. Moreover, the concept of the middleware is stretched to include the management, controlling and monitoring of the environment. The new middleware types are responsible for managing the processes of deploying resources by the providers and discovering them according to consumers' requests. As shown in Figure A.1, the core functions of middleware include [163]:

- **Deployment function:** describes the process of deploying resources.
- **Discovery function:** describes the way of discovering resources according to consumers' needs.

- **Invocation function:** describes the way of invoking the discovered resources, do the process and return the results.

All these core functions require a number of supporting functions in order to lunch in planetary-scale system. These supporting functions are [163]:

- **Uniform computing access:** makes all computational resources appear to have a uniform interface to the consumers, while there is heterogeneous in the reality.
- **Uniform data access:** is responsible for performing the semantic of the data before delivering to the consumers.
- **Authentication, delegation and source communication:** assists the system to build and manage the level of authorisation and authentication.
- **Identify certificate management:** provides the system with the policies for carrying out the management tasks.

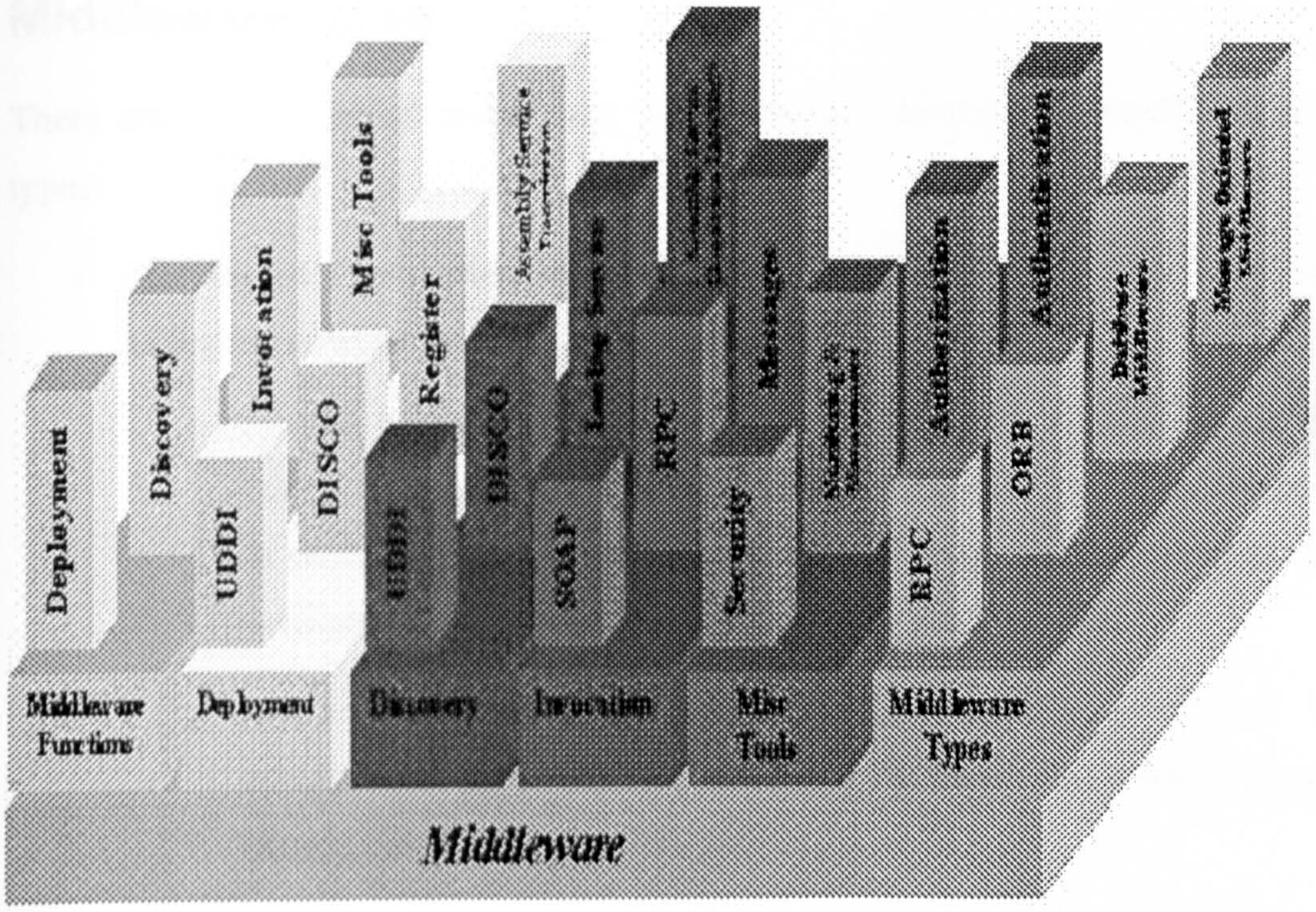


Figure A.1: Middleware System

Preliminaries

All types of middleware systems should meet the following specification [164]:

- **Ease of use**
- **Location transparency:** the consumer should not be worried regarding the deploying and discovering of the resources.
- **Message delivery integrity:** message should not be lost or duplicated.
- **Message format integrity:** message should not be corrupted.
- **Semantic message format:** message should be understood by all platforms and systems.
- **Applications integrated:** distributed applications can be integrated with others easily.

Middleware types

There are many types of middleware, which can be summarised in the following types:

- **Distributed Transaction and Messaging Middleware:** this covers variety of technologies which can be outlined as follow:
  - *Remote Procedure Call (RPC):* is described as a protocol that can be utilised by a program to request a service from other programs located in another computer in a network without needs for understanding the network and communication details [3, 114, 164].
  - *Microsoft Messaging Queuing (MSMQ):* allows applications to communicate with each other depending on using request and response queued message [4, 114, 164, 165].
  - *Distributed Transaction Processing (DTP):* is a software architecture that allows multiple application programs to share resources provided

by multiple resource managers, and permits their work to be coordinated into global transactions [114, 164, 166].

- *MQSeries:* is an IBM software family whose components are used to tie together other software applications so that they can work together [114, 167, 168].

- **Object-Oriented Middleware:** Object middleware is built on the simple concepts of calling an operation in an object that resides in another system. Instead of client and server, there is client and object [3, 114, 164]. This Type of middleware covers DOCM and CORBA [3, 114, 164, 169].

- **.Net Middleware:** The .Net platform represents an evaluation of the Components Object Model (COM) [2, 3, 114, 164, 170]. It is employed to create software components that are completely object based. .Net is developed by Microsoft to support web service technology and employee SOAP [2] as a way of exchanging messages.

- **Java world:** this covers a large number of middleware types, such as:

  - *Enterprise JavaBeans (EJB):* provides a transaction processing (TP) monitor-like environment for distributed components. The TP monitor characteristics of the EJB platform allows developers to streamline development by automatically managing the entire application environment, including transactions, security, concurrency, load balancing and failover [3, 4, 114, 150, 171].

  - *Java 2 Platform Enterprise Edition (J2EE):* Sun Microsystems (together with industry partners such as IBM) designed J2EE to hide the complexity of requesting remote resources by the thin client. They argue that J2EE simplifies application development and decreases the need for programming and programmer training by creating standardised, reusable modular components and by enabling the tier to handle many aspects of programming automatically [3, 4, 114, 172].

- *Java Messaging Service (JMS):* is an API for sending messages and events between two or more clients in a formal way [3, 4, 114, 173].
- *Java Naming and Directory Interface (JNDI):* provides a unified interface to multiple naming and directory services. As part of the Java enterprise API set, JNDI enables seamless connectivity to heterogeneous enterprise naming and directory services [3, 4, 114, 174].
- *Jini:* is a middleware developed by Sun to leverage the deployment and usability in heterogeneous system based on using register and look-up services [3, 4, 114, 150].

- **Database-Oriented Middleware:** is middleware specialist in dealing and facilitating the transactions of messages with distributed database, whether from an application or between databases [114, 175].

APPENDIX B

GANG OF FOUR-STRUCTURAL PATTERNS

Design patterns are level up from code and typically show how to simplify a problem into numbers of modules. A design pattern is a pattern-a way to pursue intent-that uses classes and their methods in an object-oriented language. Patterns are about design and interaction of objects, as well as providing a communication platform concerning elegant, reusable solutions to commonly encountered programming challenges.

The Gang of Four (GoF) patterns are generally considered the foundation for all other patterns. They are categorized in three groups: Creational, Structural, and Behavioural patterns. GoF has been a de facto reference for any Object-Oriented software developer. The creational patterns had been discussed through chapter six, while behavioural patterns are out of the scope of this thesis. The structural patterns are shown below [134, 176, 177]:

- **Adapter:** allows the system to provide a new interface for a class that already exists in order to be fitted with the users' requirements. Also it provides the tools for building a new class which will have "pluggable" adapters tailored for individual client needs. The UML class diagram for this pattern is shown below.

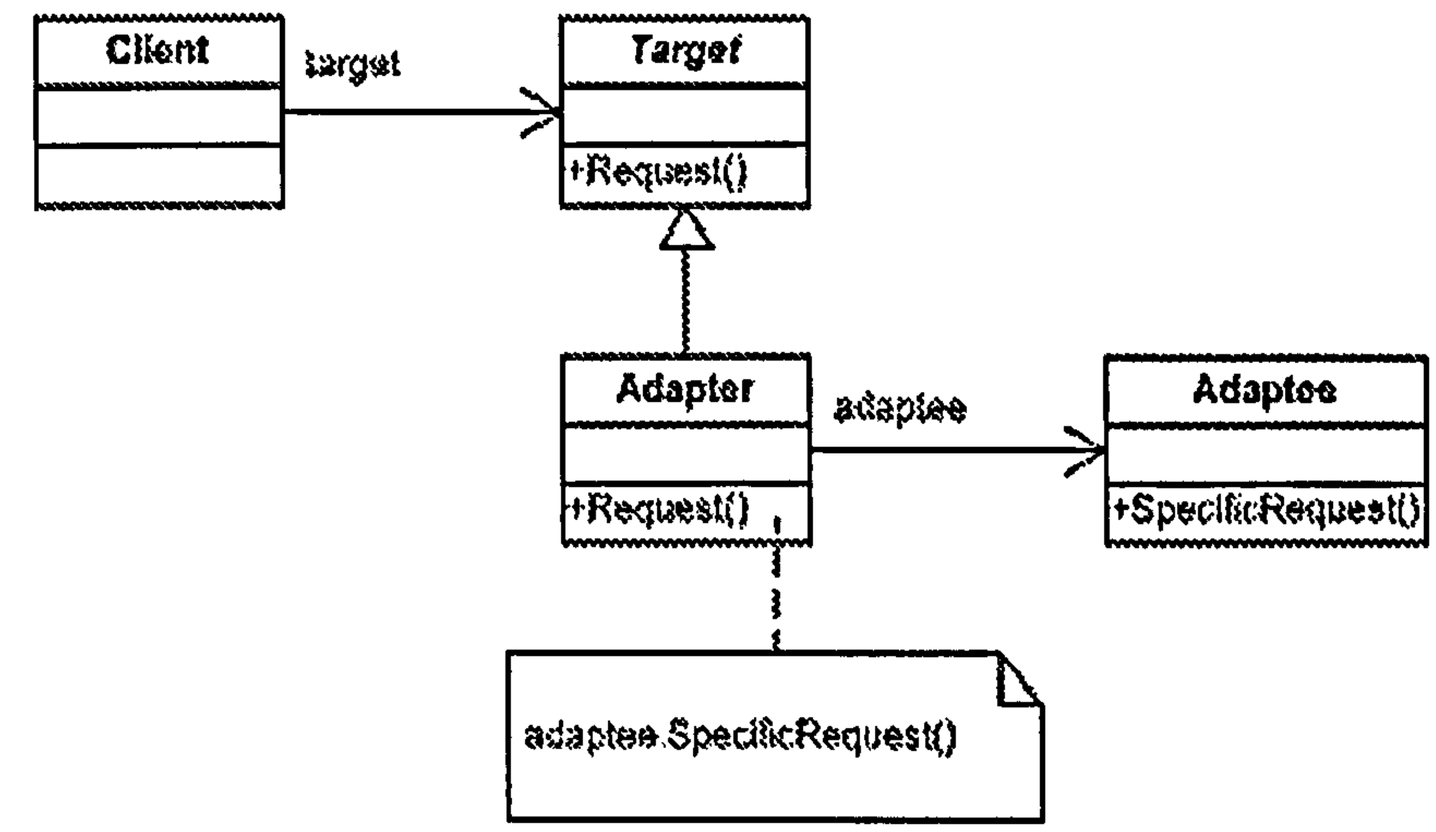


Figure B.1: Adapter Pattern [134]

- **Bridge:** decouples an abstraction from its implementation so that the two can vary independently. This means that the system allows the class and its interface to be changed independently over time which can lead to more reuse and less future shock. Moreover, this pattern authorizes us to dynamically switch between implementations at runtime allowing increased levels of runtime flexibility. Figure B.2 demonstrates UML class diagram for bridge pattern.

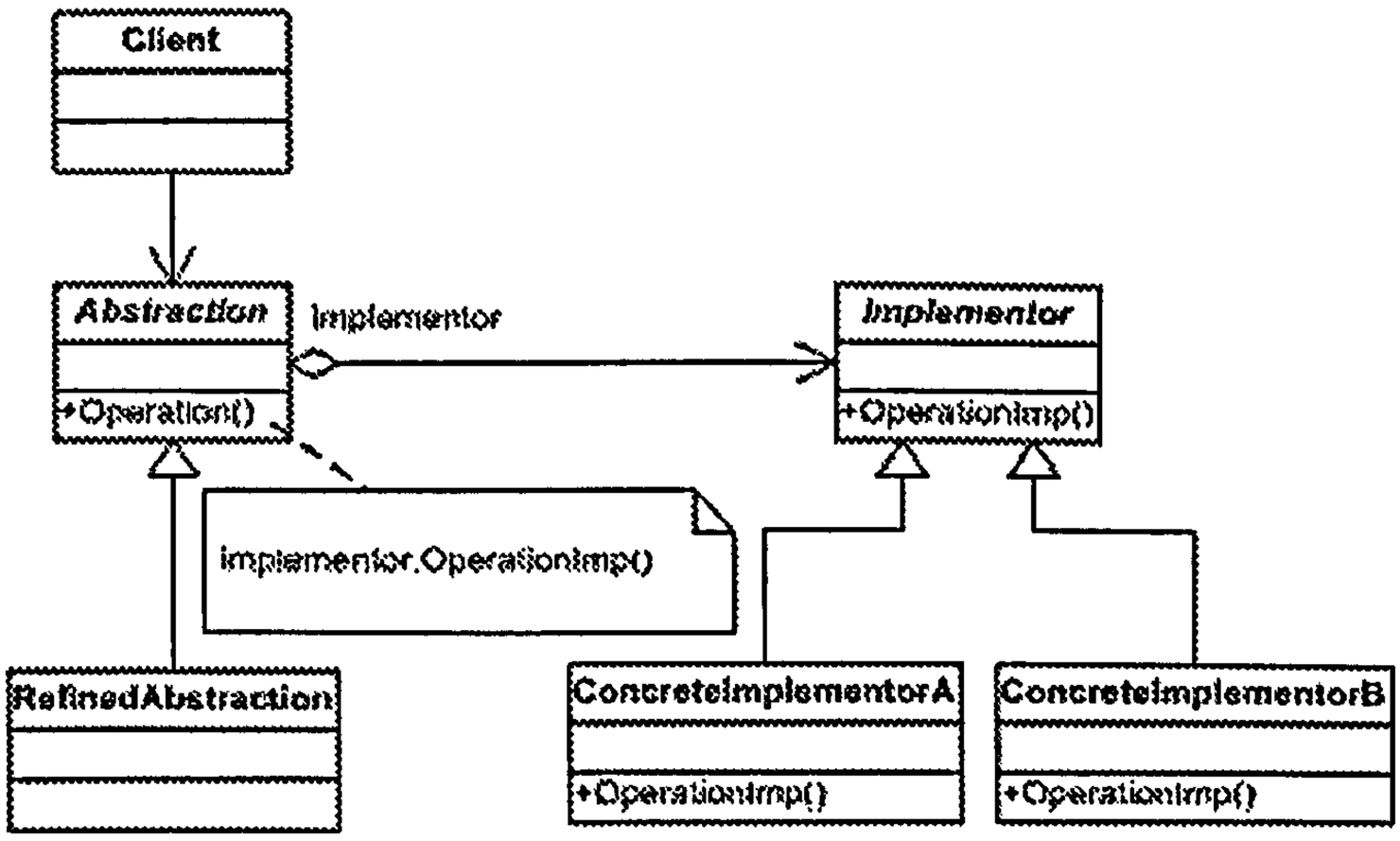


Figure B.2: Bridge Pattern [134]

- **Composite:** A flexible pattern that provides complex and flexible tree structures. The trees can be built from various types of containers or leaf nodes, and its depth or composition can be adjusted or determined at runtime. The client is simplified as it can deal with the tree as a single object, as the Composite pattern can take care of dealing appropriately with all the differing component parts. Figure B.3 illustrates the UML class diagram for composite pattern.

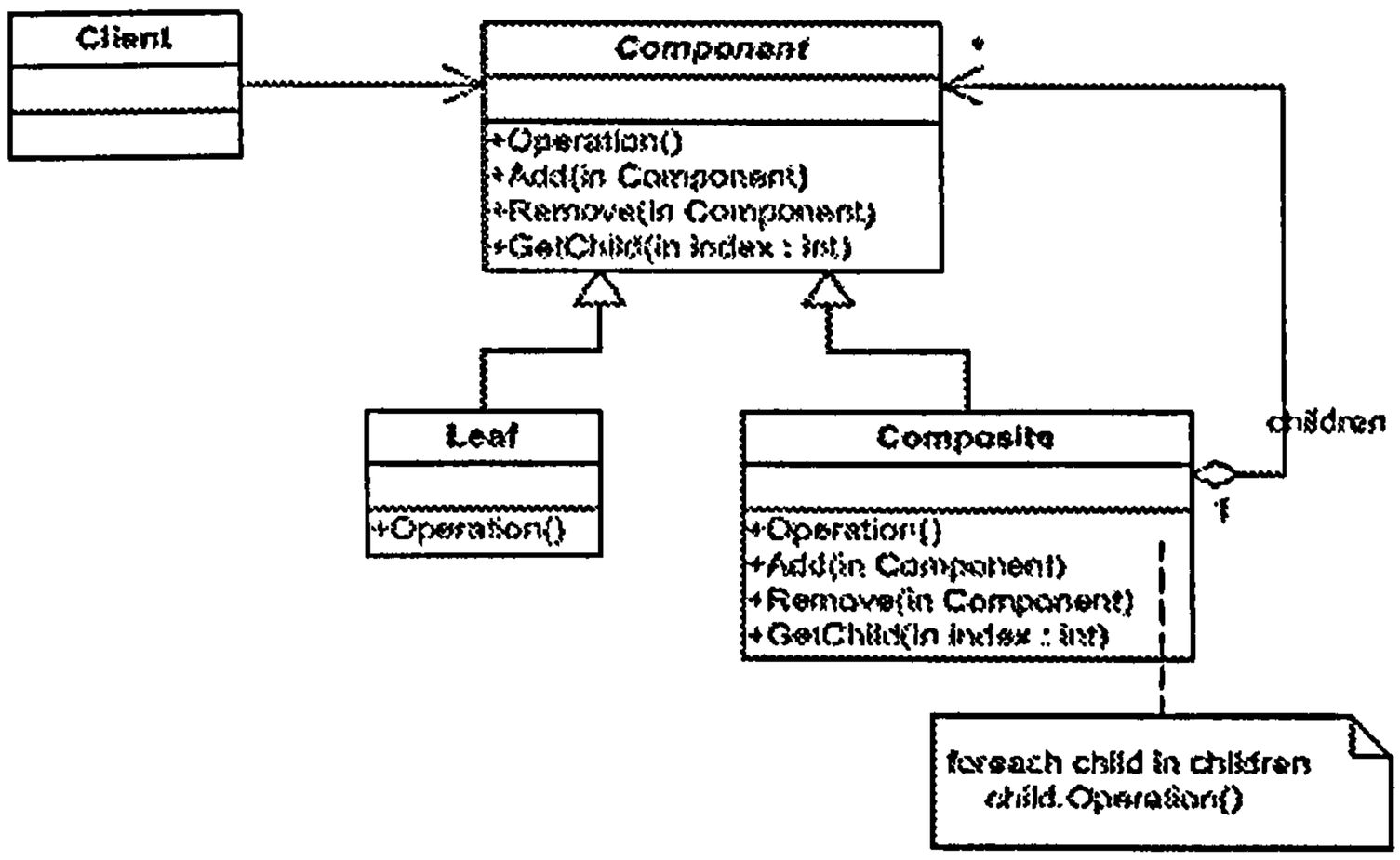


Figure B.3: Composite Pattern [134]

- **Decorator:** is utilised for dynamic object modifying at runtime by attaching new behaviours, or modifying existing ones. Decorators provide a flexible alternative to sub classing for extending functionality. Therefore, decorator pattern support the design of the system by "pay as you go" systems where overhead is incurred only when runtime, or configuration options, require it. The UML class diagram for decorator pattern is shown in Figure B.4.

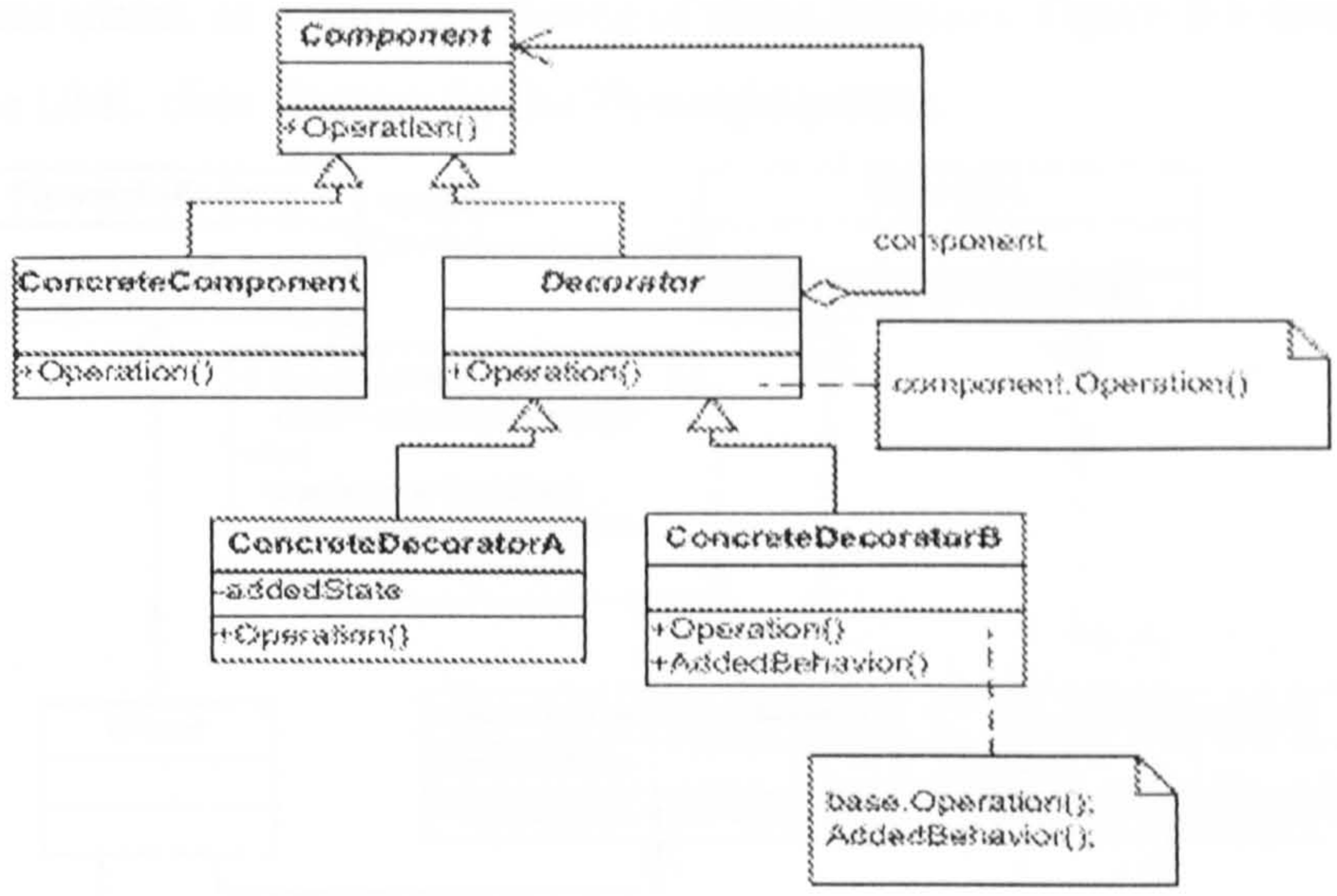


Figure B.4: Decorator Pattern [134]

- **Façade:** provides a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use. The potential benefits of this pattern are to simplify the client, compartmentalise the client, help future proof of the applications, or enable more reuse. The UML class diagram for Façade pattern is shown in Figure B.5.

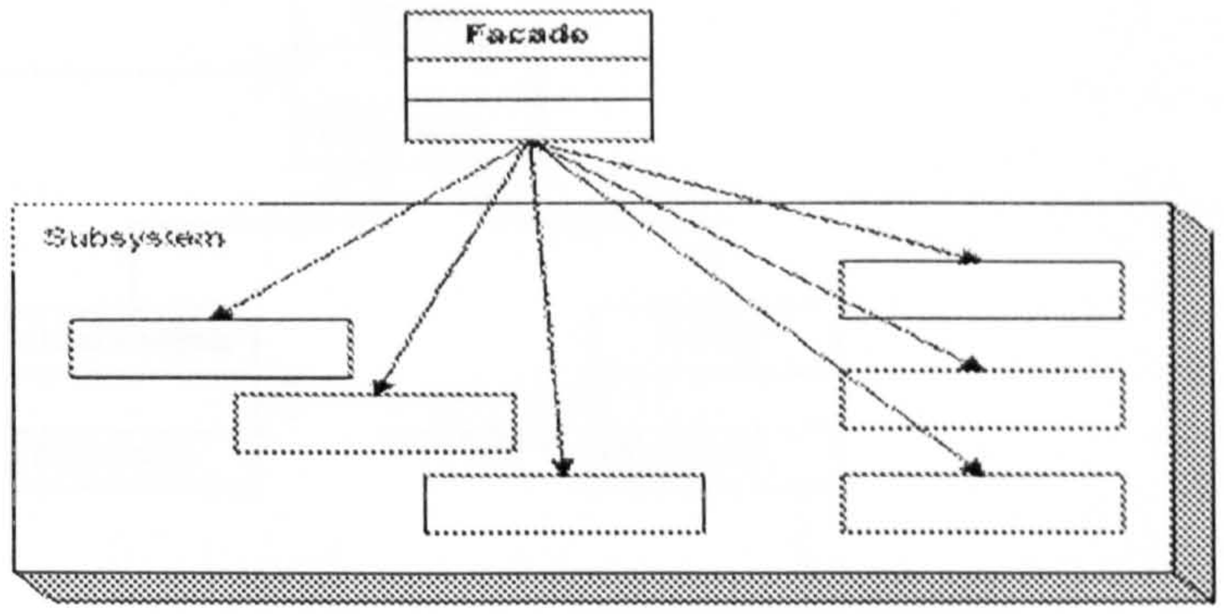


Figure B.5: Façade Pattern [134]

- **Flyweight:** uses sharing classes to support large numbers of fine-grained objects efficiently. This can be achieved by optimises memory use when there is a designed class which is demanded by huge number of clients. The pattern is most applicable if there will be clusters of runtime objects that have similar

state (data), as it arranges sharing of these instances. Figure B.6 demonstrates the UML class diagram for the Flyweight pattern.

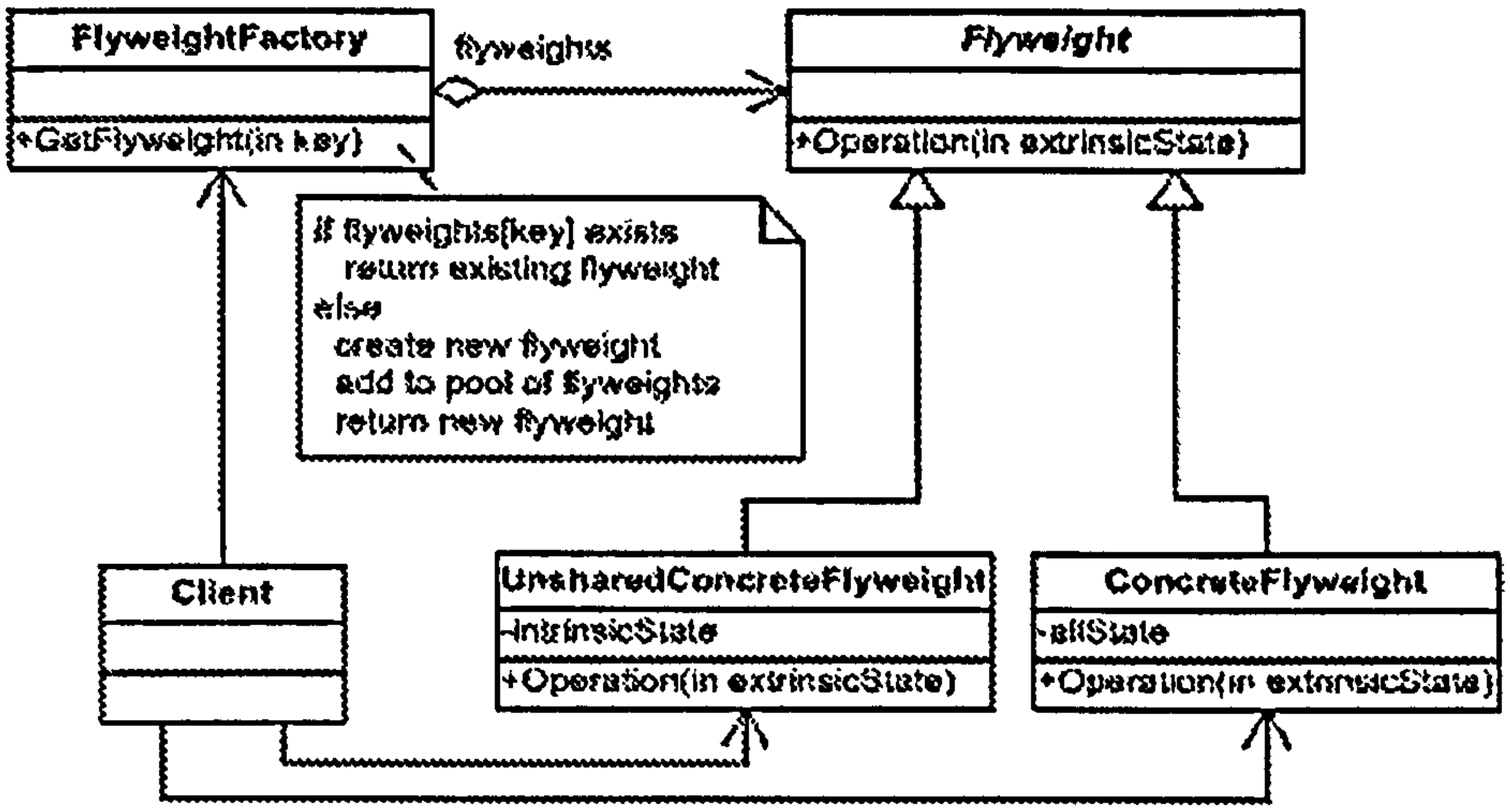


Figure B.6: Flyweight Pattern [134]

- **Proxy:** This pattern provides a surrogate object that controls access to some other object. Examples include objects upon a remote system, objects that have client authentication requirements, or objects that are expensive to fully create so some client purposes may be served with just a cut down instantiation. Figure B.7 illustrates Proxy pattern.

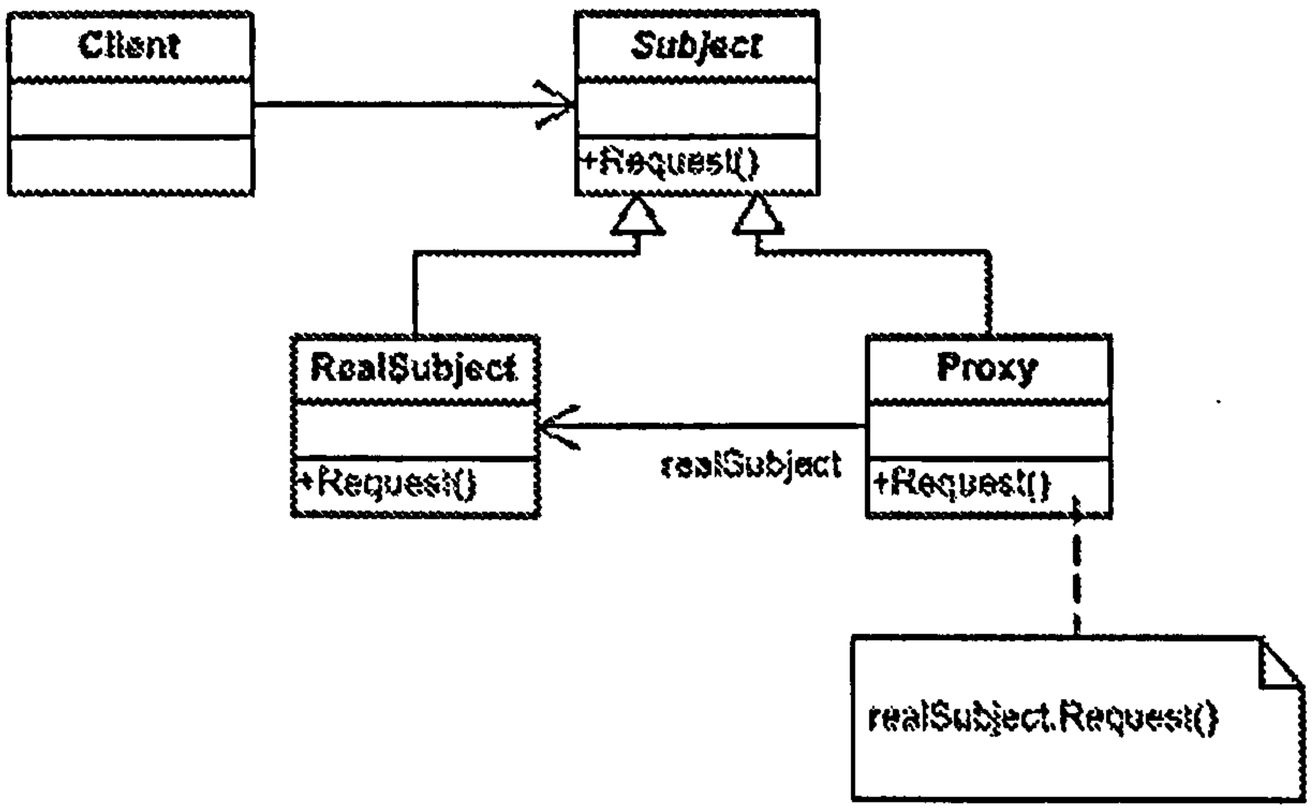


Figure B.7: Proxy Pattern [134]

APPENDIX C

LIST OF ABBREVIATIONS

Abbreviation	Description
ASIDL	Assembly Services and Infrastructures Description Language
ASIF	Assembly Services and Infrastructures Framework
BMU	Best Matching Unit
CIM	Common Information Model
DISCO	Discovery
EHMS	E-Health Monitoring System
GoF	Gang of Four
JSU	Job Schedule Unit
KNN	K-Nearest Neighbourhood
MOWS	WSDM Management of Web Services
MSDL	Monitor Session Description Language
MUWS	WSDM Management Using Web Services
ODS	On-Demand Service
OGSA	Open Grid service Architecture
QoS	Quality of Service
SADL	Sensor and Actuator Description Language
SAF	Sensors and Actuators Framework
SLA	Service Level of Agreement
SM-VSM	Self-Management Viable System Model
SOAP	Simple Access Description Language
SOM	Self Organising Map
SRU	Service Reservation Unit
SSM	Soft Systems Methodology
SVM	Support Vector Machine
UDDI	Universal Discovery, Description and Integration
UML	Unified Modelling Language
VServer	Virtual Servers
VSM	Viable System Model
WS	Web Service

WSDL	Web Service Description Language
WSDM	Web Services Distributed Management
XML	eXtensible Markup Language

APPENDIX D

PUBLICATIONS BY THE AUTHOR

1. B. Ahmad, W. Omar, A. Taleb-Bendiab. *Intelligent Monitoring Model For Sensing Financial Application Behaviour Based On Grid Computing Overlay.* in *Submitted to 2006 IEEE International Conference on Services Computing (SCC 2006).* 2006. USA.
2. M. Yu, A. Taleb-Bendiab, D. Reilly, W. Omar. *Multi-Standard Service Interoperation Protocol Through Polyarchical Middleware.* in *4th Annual Postgraduate Symposium on The Convergence of Telecommunications, Networking & Broadcasting (PGNet2003).* 2003. Liverpool, U.K.
3. M. Yu, A. Taleb-Bendiab, D. Reilly, W. Omar. *Ubiquitous Service Interoperation through Polyarchical Middleware.* in *IEEE / WIC International Conference on Web Intelligence(WI 2003).* 2003. Halifax - Canada.
4. M. Yu, A. Taleb-Bendiab, D. Reilly, E. Grishikashvili, W. Omar. *Polyarchical Middleware for On-Demand and Multi-Standard Services' Composition for Ubiquitous Computing.* in *UNITN - International Conference on Service Oriented Computing.* 2003. Trento - Italy.
5. W. Omar, A. Taleb-Bendiab, M. Yu. *An Open Standard Description Language for Semantic Grid Services Assembly for Autonomic Computing Overlay.* in *IEEE International Conference on Services Computing (IEEE SCC 2004).* 2004. SHANGHAI-CHINA.
6. W. Omar, A. Taleb-Bendiab, Y. Karam. *PlanetLab Overlay: Experimenting With Sensing and Actuation Support For Situated Autonomic Computing Services For The Planetary- Scale System.* in *iiWAS.* 2005. Malaysia.
7. W. Omar, A. Taleb-Bendiab, Y. Karam. *PlanetLab Overlay: Experimenting with Sensing and Actuation Support for Situated Autonomic Computing Services.* in *6th PG net2005 conference.* 2005. Liverpool, UK.
8. W. Omar, A. Taleb-Bendiab, Y. Karam. *A Machine Learning Middleware for On Demand Grid Services Engineering and Support.* in *Workshop on Computer Supported Activity Coordination (CSAC-2005).* 2005. MIAMI BEACH- FLORIDA-USA.
9. W. Omar, A. Taleb-Bendiab, *PlanetLab Overlay: Experimenting With Sensing And Actuation Support For Situated Autonomic Computing Services.* Submitted to International Journal of Intelligent Information Technologies (IJIIT) (accepted for extended version), 2005.

10. W. Omar, B. Ahmad, A. Taleb-Bendiab. *Grid Overlay for Remote E-Health Monitoring.* in *The 4th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-06).* 2006: IEEE Computer Society.
11. W. Omar, A. Taleb-Bendiab, Y. Karam, *Autonomic Middleware Services for Just-In-Time Grid Services Provisioning.* Journal of Computer Sciences, 2006.
12. W. Omar, A. Taleb-Bendiab. *Self-Management Viable System Model for Planetary Scale Environment (SM-VSM).* in *Submitted to 3rd IEEE Workshop on Engineering of Autonomic Systems (EASe 2006).* 2006. Columbia, MD, USA.
13. W. Omar, A. Taleb-Bendiab. *Service Oriented Architecture for Remote E-Health Monitoring System.* in *Submitted to 2006 IEEE International Conference on Services Computing (SCC 2006).* 2006. USA.
14. W. Omar , B. Ahmad, A. Taleb-Bendiab, Y. Karam. *A Software Framework for Open Standard Self-Managing Sensor Overlay For Web Services.* in *7th International Conference on Enterprise Information Systems (ICEIS2005).* 2005. MIAMI BEACH- FLORIDA-USA.

References

1. M. Chen, E. Kiciman, E. Fratkin, A. Fox, E. Brewer. *Pinpoint: Problem Determination in Large, Dynamic Internet Services*. in *International Conference on Dependable Systems and Networks*. 2002: IEEE Computer Society-Washington, DC, USA.
2. M. Thomas, M. Parihar, E. Ahmed, J. Chandler, B. Hatfield, R. Lassan, P. MacIntyre, D. Wanta, *ASP.NET Bible (Paperback)*. First ed. 2001: Wiley.
3. F. Berman, G. fox, A. Hey, *Grid Computing: Making the Global Infrastructures a Reality*. Wiley Series in Communications Networking and Distributed Systems. 2003, Chichester, West Sussex, England: John Wiley and Sons Ltd.
4. I. Foster, C. Kesselman, *Grid 2: Blueprint for a New Computing Infrastructure*. Second ed. 2004, San Francisco, USA: Morgan Kufmann.
5. Fellenstein, G., *On Demand Computing: Technologies and Strategies*. 2005: IBM press.
6. V. Berstis, L. Ferreira, *Fundamentals of Grid Computing*. 2002, IBM.
7. Badr, N., *An Investigation into Autonomic Middleware Control Services to Support Distributed Self-Adaptive Software*, in *School of Computing and Mathematical Science*. 2003, Liverpool John Moores University: Liverpool.
8. M. Yu, A. Taleb-Bendiab, D. Reilly, W. Omar. *Multi-Standard Service Interoperation Protocol Through Polyarchical Middleware*. in *4th Annual Postgraduate Symposium on The Convergence of Telecommunications, Networking & Broadcasting (PGNet2003)*. 2003. Liverpool, U.K.
9. M. Yu, A. Taleb-Bendiab, D. Reilly, W. Omar. *Ubiquitous Service Interoperation through Polyarchical Middleware*. in *IEEE / WIC International Conference on Web Intelligence(WI 2003)*. 2003. Halifax - Canada.
10. M. Yu, A. Taleb-Bendiab, D. Reilly, E. Grishikashvili, W. Omar. *Polyarchical Middleware for On-Demand and Multi-Standard Services' Composition for Ubiquitous Computing*. in *UNITN - International Conference on Service Oriented Computing*. 2003. Trento - Italy.
11. B. Ahmad, W. Omar, A. Taleb-Bendiab. *Intelligent Monitoring Model For Sensing Financial Application Behaviour Based On Grid Computing Overlay*. in *Submitted to 2006 IEEE International Conference on Services Computing (SCC 2006)*. 2006. USA.
12. W. Omar, A. Taleb-Bendiab, M. Yu. *An Open Standard Description Language for Semantic Grid Services Assembly for Autonomic Computing Overlay*. in *IEEE International Conference on Services Computing (IEEE SCC 2004)*. 2004. SHANGHAI-CHINA.

13. W. Omar, A. Taleb-Bendiab, Y. Karam. *PlanetLab Overlay: Experimenting With Sensing and Actuation Support For Situated Autonomic Computing Services For The Planetary- Scale System.* in *iiWAS.* 2005. Malaysia.
14. W. Omar, A. Taleb-Bendiab, Y. Karam. *PlanetLab Overlay: Experimenting with Sensing and Actuation Support for Situated Autonomic Computing Services.* in *6th PG net2005 conference.* 2005. Liverpool, UK.
15. W. Omar, A. Taleb-Bendiab, Y. Karam. *A Machine Learning Middleware for On Demand Grid Services Engineering and Support.* in *Workshop on Computer Supported Activity Coordination (CSAC-2005).* 2005. MIAMI BEACH- FLORIDA-USA.
16. W. Omar, A. Taleb-Bendiab, *PlanetLab Overlay: Experimenting With Sensing And Actuation Support For Situated Autonomic Computing Services.* Submitted to International Journal of Intelligent Information Technologies (IJIIT) (accepted for extended version), 2005.
17. W. Omar, A. Taleb-Bendiab, Y. Karam, *Autonomic Middleware Services for Just-In-Time Grid Services Provisioning.* Journal of Computer Sciences, 2006.
18. W. Omar, A. Taleb-Bendiab. *Self-Management Viable System Model for Planetary Scale Environment (SM-VSM).* in *Submitted to 3rd IEEE Workshop on Engineering of Autonomic Systems (EASe 2006).* 2006. Columbia, MD, USA.
19. W. Omar, A. Taleb-Bendiab. *Service Oriented Architecture for Remote E-Health Monitoring System.* in *Submitted to 2006 IEEE International Conference on Services Computing (SCC 2006).* 2006. USA.
20. W. Omar, A.T.-B., Y. Karam. *PlanetLab Overlay: Experimenting With Sensing and Actuation Support For Situated Autonomic Computing Services For The Planetary- Scale System.* in *iiWAS.* 2005. Malaysia.
21. W. Omar, B. Ahmad, A. Taleb-Bendiab. *Grid Overlay for Remote E-Health Monitoring.* in *The 4th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-06).* 2006: IEEE Computer Society.
22. W. Omar , A.T.-B., *E-Health Support Services Based on Service Oriented Architecture.* IEEE IT professional journal, March/April 2006. 8(2).
23. Lizardo, O. *Towards An Impure Sociology: Formalism, Behavioral Realism and the Interdisciplinary Challenge in Social Theory.* in *Philosophy of Science Association Meetings (PSA).* 2004. Austin.
24. White, H.C., *Identity and Control:A Structural Theory of Social Action.* Princeton University Press, 1992: p. 448.
25. F. Xu, M.H.E., D.J.Baker, and S. J. Cox. *Tools and Support for Deploying Applications on the Grid.* in *IEEE International Conference on Services Computing.* 2004. Shanghai, China: IEEE Computer Society.
26. IT:, S.B.a. and T.G.-E.A. Enterprise. 2004, HP, Intel, Oracle.
27. F. Berman, G.f., A. Hey, *Grid Computing: Making the Global Infrastructures a Reality.* Wiley Series in Communications Networking and Distributed Systems. 2003, Chichester, West Sussex, England: John Wiley and Sons Ltd.
28. I. Foster, a.C.K., *Grid 2: Blueprint for a New Computing Infrastructure.* Second ed. 2004, San Francisco, USA: Morgan Kufmann.

29. Hoschek, W., *Peer-to-Peer Grid Databases for Web Service Discovery*. 2002.
30. Shankland, S., *Grid Computing Luring Mainstream Backers*. 2002.
31. J. Kurose, K. Ross, *Computer Networking: A Top-Down Approach Featuring The Internet*. Third ed. 2005: Addison Wesley.
32. L. Ferreira, V. Berstis, J. Armstrong,M. Kendzierski,A. Neukoetter,M. Takagi,R. Bing-Wo, A. Amir, R. Murakawa, O. Hernandez, J. Magowan, N. Bieberstein, *Introduction to Grid Computing with Globus*. 2003: IBM. 290.
33. I. Foster, C. Kesselman,S. Tuecke, *The Anatomy of the Grid*. 2001.
34. G. Menkhaus, W. Pree, P. Baumeister, U. Deichsel, *Interaction of Device-Independent User Interfaces with Web services*. 2002, Software Resaerch Lab.
35. IBM, *On Demand Glossary*. 2003, IBM.
36. I. Foster, C. Kesselman, J. Nick, S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. 2002.
37. B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, M. Bowman, *PlanetLab: An Overlay Testbed for Broad-Coverage Services*. ACM Computer Communications Review, 2003. 33(3).
38. PlanetLab, *http://www.planet-lab.org*.
39. PlanetLab, *http://planetlab.millennium.berkeley.edu/*. 5-2005.
40. PlanetLab:PlanetLabUser'sGuide, *http://www.planet-lab.org/doc/UsersGuide.php*.
41. CKRM, *http://ckrm.sourceforge.net*. 8-2005.
42. Miller, B., *The Autonomic computing edge: The "Standard" way of autonomic computing*. 2005, IBM.
43. K. Herrmann, G. Mühl, K. Geihs, *Self Management: The Solution to Complexity or Just Another Problem?* IEEE DISTRIBUTED SYSTEMS ONLINE 1541-4922© 2005, 2005. 6(1).
44. G. Lanfranchi, P. Peruta, A. Perrone,D. Calvanese, *Toward A New Landscape of Systems Management in an Autonomic Computing Environment*. IBM Systems Journal, 2003. 42(1): p. 119-128.
45. A. Ganek, T. Corbi, *The Dawning of the Autonomic Computing Era*. IBM Systems Journal, 2003. 42(1): p. 5-18.
46. Murch, R., *Autonomic Computing*, ed. I. Press. 2004: Prentice Hall.
47. Tosi, D., *Research Perspectives in Self-Healing Systems*. 2004.
48. IBM, *Autonomic Computing*. 2003.
49. Partners, G.T., *The Autonomic Computing Report – Characteristics of Self Managing IT Systems*. 2002.
50. D. Chess, C. Palmer, S. White, *Security in an Autonomic Computing Environment*. IBM Systems Journal, 2003. 42(1): p. 107-111.
51. J. Kephart, D. Chess, *The Vision of Autonomic Computing*. IEEE Computer, 2003. 36(1): p. 41-50.
52. T. Cofino, Y. Doganata, Y. Drissi, T. H. Fin, M. J. Kim, L. Kozakov, M. Laker, *Towards knowledge management in autonomic systems*. 2002, IBM T. J. Watson Research Center, Hawthorne, NY 10562.
53. Paulson, L., *Computer System: Heal Theyself*. IEEE Computer, 2002. 35(8): p. 20-22.

54. N. Badr, A. Taleb-Bendiab, D. Reilly. *Policy-Based Autonomic Control Service*. in *Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2004)*. 2004. New York.
55. N. Badr, A. Taleb-Bendiab, M. Randles, D. Reilly. *A Deliberative Model for Self-Adaptation Middleware Using Architectural Dependency*. in *2nd International Workshop on Self-Adaptable and Autonomic Computing Systems (SAACS'04)*. 2004. Spain.
56. Waldrop, M., *Autonomic Computing: The Technology of Self-Management*. 2003, Woodrow Wilson International Center for Scholars.
57. M. Randles, A. Taleb-Bendiab, P. Miseldine, A. Laws. *Adjustable Deliberation of Self-Managing Systems*. in *Engineering of Computer Based Systems (ECBS 2005)*. 2005. Maryland, USA.
58. J. Clark, E. Gutentag, D. Satola, *How Technical Standards Affect E-Commerce*. 2003, American Bar Association: San Francisco, California, USA.
59. IBM, *An Architectural Blueprint for Autonomic Computing*. 2004.
60. T. Studwell, K. Sankar, J. Baekelmans, P. Brittenham, T. Deckers,C. Laet, E. Merenda, B. Miller, D. Ogle, B. Rajaraman, K. Sinclair, J. Sweitzer, *Adaptive Services Framework*. 2003, IBM.
61. B. Topol, D. Ogle, D. Pierson, J. Thoensen, J. Sweitzer, M. Chow, M. Hoffmann, P. Durham, R. Telford, S. Sheth, T. Studwell, *Automating Problem Determination: A First Step Toward Self-Healing Computing Systems*. 2003, IBM.
62. A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu, *Web Services Agreement Specification (WS-Agreement)*. Global Grid Forum (GGF), 2004.
63. S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Malhotra, A. Nadalin, N. Nagaratnam, M. Nottingham, H. Prafullchandra, C. v. Riegen, J. Schlimmer, C. Sharp, J. Shewchuk, *Web Services Policy Framework (WS-Policy)*. 2004, BEA Systems Inc., International Business Machines Corporation, Microsoft Corporation, Inc., SAP AG, Sonic Software, and VeriSign Inc.
64. S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, S. Tuecke, W. Vambenepe, B. Weihl, *Web Services Notification (WS-Notification)*. 2004, International Business Machines Corporation, Sonic Software Corporation, SAP AG, Hewlett-Packard Development Company, Akamai Technologies Inc. and Tibco Software Inc.
65. DMTF, *http://www.dmtf.org/standards/cim/*. 11-2004.
66. WSDM, *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm*.
67. WSRF, *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf*.
68. Satterthwaite, E., *Debugging Tools for High Level Languages*. Software Practice and Experience, 1972: p. 197-217.
69. Sommerville, I., *Software Engineering*. Fourth ed. 1992: Addison Wesley.

70. Rover, T. *Performance Evaluation: Integrating Techniques and Tools into Environments and Frameworks*. in *Supercomputing*. 1994. Washington, D.C., United States: IEEE Computer Society Press.
71. A. Waheed, T. Rover. *Structured Approach to Instrumentation System Development and Evaluation*. in *ACM/IEEE Supercomputing Conference (SC'95)*. 1995. San Diego California.
72. M. Heath, J. Etheridge, *Visualizing the Performance of Parallel Programs*. IEEE Software, 1991. **8**(5): p. 29-39.
73. R. Want, T. Pering, D. Tennenhouse, *Comparing Autonomic and Proactive Computing*. IBM Systems Journal, 2003. **42**(1): p. 129-135.
74. *Healthcare Information and Management Systems Society E-Health Special Interest Group: Definition of E-Health*. (2002).
75. e-Diamond, *http://e-science.ox.ac.uk/public/eprojects/e-diamond/index.xml.ID=body.1_div.1*. 6-2005.
76. InSiteOne, *http://www.Insiteone.com*.
77. 2nrich, *http://www.cms.livjm.ac.uk/2nrich/*. 6-2004.
78. Folding, *http://www.stanford.edu/group/pandegroup/folding/*. 2004.
79. PDB, *http://www.rcsb.org/pdb/*.
80. GXD, *http://www.informatics.jax.org/mgihome/GXD/aboutGXD.shtml*. 6-2005.
81. GridPhyn, *www.griphyn.org*. 6-2005.
82. LHCGrid, *http://lcg.web.cern.ch/LCG/*. 9-2004.
83. iVDGL, *www.ivdgl.org*. 2005.
84. AstroGrid, *http://www.astrogrid.ac.uk*. 6-2005.
85. VISTA, *http://www.vista.ac.uk*.
86. LaMonica, M., ***IBM draws self-management blueprint***. April 2003.
87. R. Sterritta, M. Parasharb, H. Tianfieldc, R. Unlandd, *A Concise Introduction to Autonomic Computing*. Advanced Engineering Informatics, 2005. **19**: p. 181–187.
88. A. Laws, A. Taleb-Bendiab, S.J. Wade, D. Reilly. *From Wetware to Software: A Cybernetic Perspective of Self-Adaptive Software*. in *Second International Workshop, IWSAS 2001*. 2001. Balatonfüred, Hungary: Springer.
89. S.Beer, *Diagnosing the System for Organizations*. 1985, Chichester: John Wiley & Sons.
90. A. Laws, A. Taleb Bendiab, S.J. Wade. *Genetically Modified Software: Realizing Viable Autonomic Agency*. in *2nd GSFC/IEEE Workshop on Radical Agent Concepts (WRAC'05)*. 2005. NASA GSFC Visitor's Center, Greenbelt, MD.
91. D. Garlan, B. Schmerl. *Model-Based Adaptation for Self-Healing Systems*. in *ACM SIGSOFT Workshop on Self-Healing Systems (WOSS'02)*. 2002. Charleston,South Carolina, USA.
92. P. Oriezy, M. M. Gorlick, R. N. Taylor, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, A. Wol, *An Architecture-Based Approach to Self-Adaptive Software*. IEEE Intellingent Systems, 1999. **14**.

93. Pereira, E., *Impromptu: Software Framework for Self-Healing Middleware Services*, in *School of Computing and Mathematical Science*. 2005, Liverpool John Moores University: Liverpool.
94. D. Bustard, R. Sterritt, A. Taleb-Bendiab, A. Laws, M. Randles, F. Keenan. *Towards a Systemic Approach to Autonomic Systems Engineering*. in *ECBS 2005*. 2005.
95. Beer, S., *The heart of the Interprise*. 1979: Chichester: John Weiley & Sons.
96. Beer, S., *Brain of the Firm*. 1981: Chichester: John Wiley & Sons.
97. Herring, C. *The Pattern of the Viable System and its Language*. in *KoalaPLoP 2001*. 2001. Melbourne, Australia.
98. C. Alexander, S.I., M. Silverstein, *A Pattern Language: Towns, Buildings, Constructons. Senter for Environmental Structure*. Oxford University Press. 0-19-501919-9, 1977.
99. M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, E. Brewer. *Failure Diagnosis Using Decision Trees*. in *First International Conference on Autonomic Computing (ICAC'04)*. 2004. New York, USA.
100. G. Candea, M. Delgado, M. Chen, A. Fox. *Automatic Failure-Path Inference: A Generic Introspection Technique for Internet Applications*. in *Third IEEE Workshop on Internet Applications*. 2003.
101. M. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox , E. Brewer. *Path-Based Failure and Evolution Management*. in *Network System Design & Implementation (NSDI '04)*. 2004.
102. V. Kumar, B. Cooper, K. Schwan. *Distributed Stream Management using Utility-Driven Self-Adaptive Middleware*. in *International Conference on Autonamic Computing (ICAC)*. 2005. New jersy USA.
103. G. Blair, G. Coulson, P. Grace. *Research Directions in Reflective Middleware: the Lancaster Experience*. in *Workshop on Adaptive and Reflective Middleware*. 2004. Canada.
104. O. Babaoglu, M. Jelasity, A. Montresor. *Grassroots Approach to Self-Management in Large-Scale Distributed Systems*. in *EU-NSF Strategic Research Workshop on Unconventional Programming Paradigms*. 2004. Mont Saint-Michel, France.
105. J. Joseph, M. Ernest, C. Fellenstein, *Evolution of Grid Computing Architecture and Grid Adoption Models*. IBM Technical Journal, 2004.
106. Sun, *Grid-Computing Architecture for Design Automation in Higher Education*. 2004, Sun MicroSystem.
107. J. Joseph, M.E., C. Fellenstein, *Evolution of Grid Computing Architecture and Grid Adoption Models*. IBM Technical Journal, 2004.
108. Krill, P., *Web Services Distributed Management spec approved*. 2005, InfroWorld.
109. Tuck, J., *Practical Application of the Web Services Distributed Management Standard*. 2004, Web Service Journal.
110. McKendrick, J., *IBM Puts WSDM Behind IT Automation*. 2005, ZDnet.
111. Travis, B., *FoodMovers: Building Distributed Applications using Visual Studio .NET*. 2003, Architag International Corporation.

112. D. Lee, J. Dongarra, S. Ramakrishna, *visPerf: Monitoring Tool for Grid Computing*. 2003, University of Tennessee and Kwangju Institute of Science and Technology.
113. D. Reilly, A. Taleb-Bendiab, *Dynamic Instrumentation for Jini Applications*. 2002.
114. Myerson, J., *The Complete Book of Middleware*. 2002, USA: Auerbach.
115. N. Satoshi, S. Mitsuhisa, *Design and implementations of nimf: towards a global computing infrastructure*. 1999.
116. Globus, *Globus Heartbeat Monitor*. 2003, http://www.globus.org/hbm/heartbeat.
117. B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, M. Thompson. *A Monitoring Sensor Management System for Grid Environments*. in *Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC-9 '00)*. 2000. Pittsburgh, Pennsylvania, USA.
118. K. Getchell, A. Miller, C. Allison. *Experiences using PlanetLab to Evaluate Network Applications*. in *6th PG net2005 conference*. 2005. Liverpool, UK.
119. M. Callachan, K. Getchell, A. Miller, C. Allison. *Using Virtual Networks for Visual Network Topology Discovery*. in *6th PG net2005 conference*. 2005. Liverpool, UK.
120. CoMon, *http://comon.cs.princeton.edu/*. 5-2005.
121. CoMon, *http://codeen.cs.princeton.edu/*. 5-2005.
122. IPREF, *http://www.planet-lab.org/logs/iperf/*. 5-2005.
123. IRISLOG, *http://www.intel-iris.net/irislog/irislog.php*. 5-2005.
124. NodeList, *http://www.planet-lab.org/db/nodes/nodelists.php*. 5-2005.
125. Trumpet, *http://jabber.services.planet-lab.org/*. 5-2005.
126. SWORDRD, *http://www.swordrd.org/*. 5-2005.
127. M. Massie, B. Chun, D. Culler, *The Ganglia Distributed Monitoring System: Design, Implementation, and Experience*. Parallel Computing, 2004. **30**(7).
128. Y. Chen, D. Bindel, H. Song, R. Katz, *An Algebraic Approach to Practical and Scalable Overlay Network Monitoring*. ACM SIGCOMM, 2004.
129. D. Karuppiah, Z. Zhu , P. Shenoy , E. Riseman. *A Fault-Tolerant Distributed Vision System Architecture for Object Tracking in a Smart Room*. in *Computer Vision Systems: Second International Workshop, ICVS2001*. 2001. Vancouver, Canada.
130. K. Herrmann , G.M.a.K.G., *Self Management: The Solution to Complexity or Just Another Problem?* IEEE DISTRIBUTED SYSTEMS ONLINE 1541-4922© 2005, January 2005. 6(1).
131. Joachims, T., *Text Categorization with Support Vector Machine: Learning With Many Relevant Features*. ECML-98. 10th European Conference on Machine Learning, Heidelberg, Germany., 1998.
132. E. Gamma, R.H., Ralph Johnson, John Vlissides, ed. *Design Patterns*. First ed. 1995, Addison-Wesley Professional.
133. Vlissides, J., *GoF à la Java*. 2001.
134. GoF, *http://www.dofactory.com*. 2-2005, Data & Object Factory.
135. Beer, S., *Diagnosing the System for Organizations*. 1985, Chichester: John Wiley & Sons.

136. Beer, S., *The heart of the Interprise.* 1979, Chichester: John Weiley & Sons.
137. U. Heuser, J. Goppert, W. Rosenstiel, A. Stevens. *Classification of Human Brain Waves using Self-Organizing Maps.* in *INTELLIGENT DATA ANALYSIS IN MEDICINE AND PHARMACOLOGY (IDAMAP).* 1996. Budapest, Hungary.
138. N. Schraudolph, T.S. *Competitive Anti-Hebbian Learning of Invariants.* in *NIPS.* 1991. Denver, Colorado, USA.
139. White, R. *Competitive Hebbian Learning 2: an Introduction.* in *European Symposium on Artificial Neural Networks (ESANN'1999).* 1999. Belgium.
140. J. Vesanto, J. Himberg, E. Alhoniemi, J. Parhankangas, *Self-Organizing Map in Matlab: the SOM Toolbox.* In proceedings of the Matlab DSP Conference 1999, Espoo, Finland, 1999: p. pp. 35-40.
141. Vesanto, J., *Using SOM in Data Mining.* Thesis for the degree of Licentiate of Science in Technology, Supervisors: Professor Olli Simula, Professor Samuel Kaski, Espoo, Finland . 2000.
142. D. Levine, T.K., M. Berenson, *Business Statistics.* Second ed. 2000: Prentice-Hall, Inc.
143. OHRING, G., *Application of Stepwise Multiple Regression Techniques to Inversion of Nimbus "IRIS" Observations.* Monthly Weather Review, 1972. **100**(5): p. 336–344.
144. V. Berstis, L.F., *Fundamentals of Grid Computing.* 2002, IBM.
145. J.U.a.M., H., *A visual tour of Open Grid Services Architecture.* 2003, IBM.
146. D. Groebner, P.S., P. Fry, K. Smith, *Business Statistics : A Decision Making Approach.* Sixth ed. 2005, New Jersey, USA: Pearson Prentice Hall.
147. J. Milton, a.J.A., *introduction to Probability and Statistics: Principles and Applications for Engineering and the Computing Sciences.* Third ed. Probability and Statistics. 1995: McGraw-Hill, Inc.
148. T. Bellwood, L.C., D. Ehnebuske, A. Hately, M. Hondo, Y. L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, *UDDI Version 3.0. UDDI Special Technical Committee Specification.* 19 July 2002.
149. UDDI, *http://www.uddi.org/specification.html.*
150. W. Edwards, T. Rodden, *Jini Example By Example.* 2001: Prentice Hall PTR.
151. WD, *http://www.w3.org/TR/2004/WD-ws-chor-model-20040324/.* 2005.
152. R. Chinnici, M. Gudgin, J. Moreau, S. Weerawarana, *Web Services Description Language (WSDL) 1.2.* 2002.
153. CIM, *http://www.dmtf.org/standards/cim/.* 8-2005.
154. J. McCann, M. Huebscher. *Evaluation Issues in Autonomic Computing.* in *GCC 2004 Workshops.* 2004: Springer-Verlag Berlin Heidelberg.
155. KNN, *http://www.cs.toronto.edu/~delve/methods/knn-class-1/knn-class-1.html.* 10-2004.
156. GridBus, *www.gridbus.org.* 2005.
157. C. Hsu, C. Chang, C. Lin, *A Practical Guide to Support Vector Classification.* 2003.
158. J. Principe, N. Euliano, W. Lefebvre, *Neural and Adaptive Systems: Fundamentals through Simulations.* 1999: Wiley.
159. Fausett, L., *Fundamentals of Neural Networks.* First ed. 1994: Prentice Hall.

160. Goldszmidt, N.F.a.M., *Learning Bayesian Network from Data.* 1998, SRI International.
161. B. Baesens, M. Egmont-Petersen, R. Castelo, J. Vanthienen. *Learning Bayesian network classifiers for credit scoring using Markov Chain Monte Carlo search.* in *Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02).* 2002: IEEE Computer Society.
162. McKendrick, J., *IBM Puts WSDM Behind IT Automation.* July 2005, ZDnet.
163. W. Johnston, J. Brooke, *Core Functions for Production Grids.* 2002, Global Grid Forum.
164. Britton, C., *IT Architectures and middleware: Strategies for Building Large, Integrated Systems.* 2001: Addison-Wesley.
165. MSMQ, *http://www.microsoft.com/windows2000/technologies/communications/msmq/default.mspx.* 12-2003.
166. DTP, *http://publib.boulder.ibm.com/infocenter/db2help/index.jsp?topic=/com.ibm.db2.udb.doc/admin/c0004558.htm.* 2005, IBM.
167. MQSeries, *http://www.mqseries.net/.* 5-2005.
168. MQSeries, *http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci214524,00.html.* 1-2004.
169. Emmerich, W., *OMG/CORBA: An Object-Oriented Middleware.* 2002, John Wiley & Sons. p. 902-907.
170. .NET, *http://www.microsoft.com/net/default.mspx.* 12-2002.
171. EJB, *http://java.sun.com/products/ejb/.* 3-2004.
172. J2EE, *http://java.sun.com/j2ee/index.jsp.* 10-2003.
173. JMS, *http://java.sun.com/products/jms/.* 3-2004.
174. JNDI, *http://java.sun.com/jndi.* 3-2004, Sun.
175. DBOM, *http://expertanswercenter.techtarget.com/eac/knowledgebaseAnswer/0,295199,sid63_gci984423,00.html.* 1-2005.
176. Metsker, S., *Design Patterns Java Workbook.* Software Patterns Series. 2002: Addison-Wesley Workbook.
177. E. Gamma, R. Helm, R. Johnson, J. Vlossodes., *Design Patterns - Elelments of Reusable Object-Oriented Software.* First ed. 1995: Addison-Wesley.