Autonomic Management of Smart Spaces

Martin Feeney, Richard Frisby

TSSG, WIT, Cork Road, Waterford, Ireland

{mfeeney, rfrisby}@tssg.org

Abstract. Management complexity of Smart Space environments increases with advances in pervasive computing domains. Extensive research work has been carried out to demonstrate the power of pervasive computing across a wide spectrum of domains. This research shows the significance of pervasive computing and it also highlights the complexity issues in the development, deployment and maintenance of such environments.

To allow industry to adopt pervasive computing, especially its offspring Smart Spaces, a method of management must be devised. While research is being carried out in this area, a new and exciting area has emerged known as Autonomic Computing. Autonomic Computing has now become an umbrella term for computer environments that display forms of autonomic control over themselves.

This paper explores the overlap of Pervasive Computing and Autonomic Computing with particular reference to Smart Spaces. It shows how the self-administrating properties of an autonomic computing solution, using IBM's Autonomic Toolkit, complements a Smart Space enabled framework such as Open Service Gateway initiative (OSGi). The conclusion reached is that the convergence of Autonomic Computing and Pervasive Computing research areas could provide a more promising future for the management of pervasive environments resulting in a reduction in cost and complexity.

1 Introduction

For pervasive computing environments, such as Smart Spaces, to become invisible to the user the need for human intervention must be reduced. Problem detection, analysis, solution planning and deployment must become background activities with little or no manual administration. It is universally known that trawling through directories of unformatted log files, to find the root of a problem, is detested by both developers and administrators alike. Once the problem is detected, the deployment of the solution has to be managed and monitored pending completion. If services start to degrade, for example due to overloaded resources, human intervention is necessary to reconfigure the service to restore it to an acceptable performance level. Software upgrades and the migration of pervasive services can be very tedious, time consuming and costly. Consequently, skilled administrator supervision is required at all times. Smart Spaces need to constantly monitor and tweak the configuration of their environment in a transparent and automated manner.

A user will not accept a Smart home if the user needs to be aware of the internal workings of their security, lighting and heating system in the case of a system crash. Having to call upon an expert to resolve every issue will also not be acceptable. Pervasive computing environments can be defined as the product of embedded computers, as well as their services and context aware information provided by an array of sensors [1]. Such an environment provides vast amounts of appropriate information and actions to aid a user in a specific task. The management of Smart Space frameworks, both local and global, could reap great benefits from autonomic concepts. Environments, such as in-house location based services and location based services, on a global scale could possibly be managed with the use of Autonomic Computing.

1.1 Autonomic Computing

The Oxford Dictionary definition of "autonomic" is an adjective relating to the part of the nervous system that controls involuntary bodily functions such as circulation and digestion. Autonomic systems manage themselves based on high-level administration objectives. "Autonomic is derived from the autonomic nervous system that governs our heart rate and body temperature thus freeing our conscious brain from dealing with these and more low level, yet vital, functions"[2]. Autonomic computing is a relatively new concept and its goal is the development of systems that are self-healing, self-configuring, self-protecting and self-optimizing in response to user/system needs [3]. These concepts are often referred to as the 'four selfs' in autonomic computing literature.

1.2 Overview

Section two, of this paper, discusses the proposed Autonomic Smart Space concepts. This section will not discuss any specifications, only an outline of the overlay between Smart Spaces and Autonomic Computing. Section three will describe the basic functionality of the Open Service Gateway initiative (OSGi) framework and also provide a brief description of the relevant components. It also introduces the main components of IBM's autonomic toolkit. Section four provides a description of an Autonomic Smart Space Architecture which combines the IBM Autonomic Architecture and the OSGi framework. Section five addresses future work and limitations of this Autonomic Smart Space Architecture.

2 Proposed Autonomic Smart Space Concepts

Smart Space environments are inherently dynamic due to the need for seamless human interaction which leads to various devices and communication modes within a Smart Space. For these devices to interact seamlessly with each other and humans, it is apparent that they and their underlying frameworks need to attain a high level of autonomic characteristics. If Smart Space homes, such as the Gator Tech Smart House [4], are to become more accessible to the standard end user, there needs to be a more hands off method of managing them. A major goal in the management of Smart Spaces is the realisation of software, systems and services that address composition, scalability, reliability and robustness as well as autonomous self-adaptation within a Smart Spaces complex environment [5]. This section shows how Autonomic Computing concepts can map to the management concepts of a Smart Space environment. The scope of this paper, and accompanying implementation, focuses on the self-healing aspect of Autonomic computing in smart spaces.

2.1 Management Challenges

There are a number of challenges that need to be dealt with when realising Smart Spaces. To allow smart spaces and smart space users to experience ubiquitous computing concepts, some key management requirements need to be addressed [6]. A highly dynamic smart space manager is needed to provide the embedded devices with self-managing capabilities such as self-improving and self-curing. A management framework for smart spaces will face major interoperable and integration challenges in combining adaptive user services with adaptive resource management in a heterogeneous environment. A software architecture needs to be defined to provide a runtime environment for smart space services to execute and provide platform independent interaction. Service bundles need to be defined and managed in such away that service adaptation and composition can be achieved by operators. The Smart Space management framework needs to allow for the creation and management of procedures that users can define. Finally information needs to be gathered, structured and made readily available as context information for users. The above management challenges need to be addressed to provide a self-managing Smart Space that will require little or no human interaction.

2.2 Solution Concepts

Self-management is the key phrase in Autonomic Computing under which falls self-configuration, self-optimization, self-healing and self-protection. Smart Space management requirements can be addressed by these 'four self's'.

Configuration within Smart Spaces is an essential function. Self-configuration in the autonomic domain necessitates a move from a singular configuration management system to a more generic policy based management system to allow for heterogeneous devices and protocols. Policy based configuration management provides a higher level of abstraction. New diverse resources may enter or leave a Smart Space frequently and need to be configured to their new space. If a Smart Space resource should leave unexpectedly, dependent resources need to be reconfigured to discover a resource with similar properties. Some of this intelligence can be built into the resource itself but a more global view on the state of the system is needed for self-configuration without consequence for other components in the system. With autonomic self-configuration, configuration can be derived from high level polices to support such an environment, thus providing a scaleable environment with the configuration power to provide composed services.

A Smart Space should be constantly striving to achieve an optimum performance level. Self-optimising components can tune themselves to provide optimal performance. Self-optimisation includes resource reallocation. For example: a video conference taking place in a smart office should be utilising the highest broadband service available to it to provide the user with the clearest possible video and voice stream. This optimisation overlap with Autonomic Computing should provide an autonomous self-adapting and reliable Smart Space, while meeting the requirements of a Smart Space Environment that wishes to make efficient use of the available resources.

Self-healing components within an autonomic environment can sense a system breakdown and deploy counteractive actions based on policies. Self-healing behaviour is a very important aspect within Smart Spaces management. Users interact with Smart Space services thus real time management is necessary to prevent system failure. Therefore, Smart Spaces have to be self-healing to provide a robust and reliable environment. The self-healing aspect of Autonomic Computing, once again, can clearly be mapped to management of pervasive computing environments.

Data, information and resources within Smart Spaces should be protected at all times from unwanted and unauthorised interference. Autonomic self-protection provides security for this area in a Smart Space. The Smart Space should be able to establish protective measures to ensure a robust environment.

The proposed Autonomic Smart Space concept mappings should provide a full autonomic management life cycle, from Smart Space service logging and analysing right through to solution planning and deployment. The autonomic cycle will start with services which reside in a Smart Space, which can be deployed by any third party vendor once the supporting framework specification is followed. The framework is assigned a remote management agent; the services can then be managed autonomically. In the case of state changes, within this dynamic environment, self-configuration, self-healing, self-optimising, self-protecting can take place based on reported state changes and available knowledge. Autonomic management within a Smart Space could be provided with autonomic concepts such as fault detection, problem analysis, solution preparation, and solution deployment using a knowledge repository and context manager.

3 Technology Overview

OSGi and the IBM Autonomic Toolkit were used to explore the overlap of the Autonomic Computing domain and the Smart Spaces domain.

3.1 Overview of OSGi Framework

"The OSGi technology is designed to ease the development of new and exciting services and applications for the latest generation of networked devices"[7]. The OSGi Alliance is an open standards organization created by Sun Microsystems, IBM, Ericsson and others.

Introduction to OSGi Framework. OSGi is a Java based service platform that can be remotely managed. OSGi enabled devices on a network allow for the management of their life-cycle from anywhere in the network. A set of one or more services run on the Service Platform and communicate with a Remote Manager to provide management. This set of remote management services provides a solution deployment technique for the Smart Space environment. A number of management bundles have been predefined for the framework including Device Access, Package Admin, Log and Permission Admin. These management bundles have Administration Permission and can control the life-cycle and configuration of other bundles on the service platform.

Framework Functionality. The OSGi framework can be divided up into five layers as identified below:

- Security Layer
- Module Layer
- Life-cycle Layer
- Service Layer
- Actual Services

The Security Layer was developed in accordance with Java2 [8] security. If security checks are required to be carried out they must conform to Java2 security. This provides the infrastructure to allow applications to be deployed and managed in a secure fashion. The security layer is intertwined with all other layers of the OSGi framework.

The Module Layer is required in order to accommodate the lack of support for packaging, deployment and validation of Java service components on the standard Java Platform. The OSGi Framework provides a generic and standardised solution for Java modularization.

The Life-cycle Layer provides a life-cycle Application Program Interface (API) for bundles. This API provides the way in which bundles are started and stopped. The Life-cycle Layer is dependant on the Module Layer and the Security Layer is optional.

The Service Layer supplies a programming model for Java bundle developers, which simplifies the development and deployment of service bundles by de-coupling the service's Java interface from its implementations.

Conclusion. In light of the above overview, of the OSGi framework, the advantages of using this framework in the development of an Autonomic system are highlighted below:

- Bundles are extensible after deployment
- Changes can be made to the framework without restarting it
- It enables easy deployment of bundles
- Features can be added to the framework
- The framework, itself, is a bundle and can be easily reconfigured
- A Logging bundle has been predefined
- A Remote Management framework is predefined

3.2 Overview of IBM Autonomic Toolkit

The IBM Autonomic Computing Toolkit is a starter kit for autonomic development. It contains technologies, tools, examples, scenarios, and documentation [9].

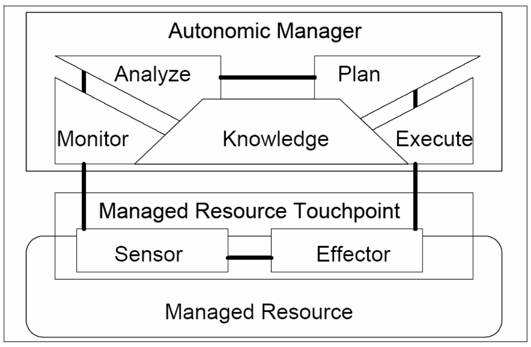

Fig. 1. Autonomic Computing Life Cycle and Control Loop [9]

Introduction to IBM's Autonomic Computing. Figure one represents IBM's autonomic vision of the autonomic life-cycle and control loop. The managed resource is some controlled system element(s), e.g. Smart Space(s), which in turn is controlled by sensors and effectors. Sensors provide the means to gather information about the state/state-changes of a managed resource. The effectors are mechanisms that have the ability to change the state of a managed resource. Fig. 1 also illustrates how sensors and effectors come together to form a management interface known as a Managed Resource Touchpoint. The Managed Resource Touchpoint enables problem detection and solution deployment within the autonomic life cycle.

The Autonomic Manager (AM) is at the core of the IBM Autonomic Computing approach and consists of a control loop including a Monitor element for information retrieval, an Analysis element to analyse gathered information and a Plan element to create a plan of action in accordance with the analysed information. This plan of action is then deployed via effectors using an Execute element. All the elements, within the AM, work in conjunction with a knowledge base of the current system layout and possible issues that could arise. The key is that the Knowledge element should grow with every new system problem and solution. The AM provides an implementation mechanism for the 'four self's' and overlaps satisfactorily with the needs of a management agent for a Smart Space environment as discussed in section 2.1.

Main Components of the IBM Autonomic Toolkit. IBM's autonomic concepts can be divided into four areas:

- Problem Detection
- Problem Analysis
- Solution Planning
- Solution Deployment

The IBM autonomic toolkit [10] provides a means to implement solutions for these four areas. The four areas cover the life-cycle of an autonomic solution to managing a Smart Space environment.

Problem Detection The first step to enabling a managed resource is the generation of state information by this resource. This can be in the form of a log file or some sort of event record, which is provided by the OSGi log specification. For this information to be useful there has to be a standard log format to stop unnecessary creation of unique management components. The generation of Common Based Events (CBE) [11] provide this and allow proprietary Smart Space components to be monitored with minimum effort. The CBE model was developed by IBM and it has been approved by the OASIS [12] standards body. IBM undertook a study of thousands of logs from different applications to produce a common format.

The next step in achieving a managed resource is the development of managed resource touchpoints. The IBM toolkit identifies four interaction styles between managed resources and their autonomic manager:

- Sensor retrieve-state

- Sensor receive-notification
- Effector perform-operation
- Effector call-out-request

Problem Analysis / Solution Planning The control loop is the next step in developing a managed resource. The key to the AM, illustrated in Figure one, is in the Resource Model [9]. The AM works on the principle of managing many autonomic applications using their corresponding resource models with one Autonomic Management Engine (AME). This will allow for many Smart Spaces to be managed by one AM. Resource models provide a structured model on how resources should be monitored. They describe the data to be collected and the logic algorithm to be used in analyses of data regarding managed resources.

The receive-notification Sensory Touchpoint that IBM has implemented in their toolkit exists as part of the resource model. The resource model receives information on a monitored resource via interfaces such as CIM (Common Information Model)/WMI (Windows Management Instrumentation), which are called providers. A set of CIM classes form a very important part of the resource model. The CIM classes are responsible for:

- The description of the resources to be monitored,
- The available properties of each resource, and
- Implementation to collect the data to be monitored.

The AME acts as a CIM Manager and provides a platform for CIM classes to run on, along with scripts provided by the resource model. The resource models can define and provide the logic for the monitoring, analysis, plan and execute phases of the autonomic control loop, but this does not allow for dynamic knowledge representation, which is needed to express true autonomic behaviour.

Solution Deployment The aim of the solution installation architecture [13] is not to fashion a single installation program that can deal with solution deployment in a singular environment, but rather an architecture that can deliver stable solution deployment in an environment that may have many component and hardware dependencies. According to IBM this type of architecture can be achieved by:

- Standardisation of the XML description of solution deployment components, which includes dependencies, file location, configuration information and actions to be performed.
- Providing a run time that can parse these definitions and carry out the necessary checks before carrying out any changes.
- Ensuring stability of a working environment by foreseeing the after effects of a new installation.
- Providing an environment that will allow communication to a host via web services to perform change management operations.
- Providing a registry for logging and tracking installations and complex relational information.

Conclusion. The IBM Autonomic Computing initiative and accompanying toolkit described above is unique and was selected for its advances in the area. It provides self-managing components, tools, scenarios and documentation that enable the realisation of an Autonomic Smart Space. These components, tools, scenarios and documentation can be divided up into four main categories - Problem Detection, Problem Analysis, Solution Planning and Solution Deployment.

4 Autonomic Smart Space Architecture

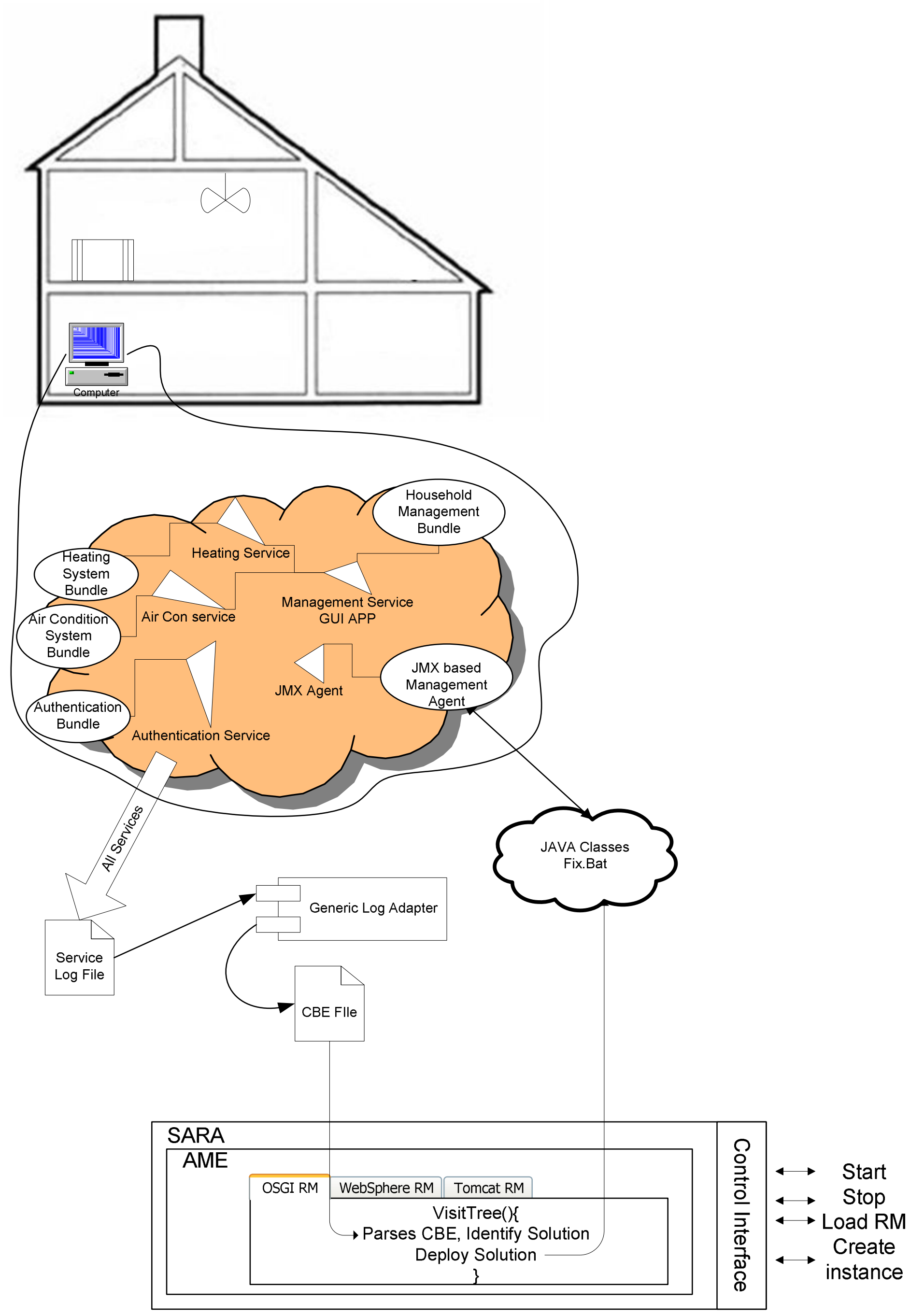

Fig. 2. Scenario Overview

4.1 Architectural Overview

The primary resources used to develop this Autonomic Smart Space architecture are the IBM Autonomic Toolkit and the OSGi framework. OSGi provides a long term vision of evolution and interoperation for the Smart Space future [14]. It provides platform and vendor independence and supports the dynamic detection and intercommunication of services along with remote management as established in section three. The IBM Toolkit provides the means to manage the OSGi framework. It provides software, tools and scenarios to work from and it is the first available toolkit of its type.

4.2 Detailed View

The basic scenario on which the Autonomic Smart Space architecture has been tested is a home management system as depicted in Fig. 2. It consists of authentication and environmental control services, some of which are co-dependent on other internal services. For example, the environmental control service is dependent on both the heating and air conditioning services. Others are dependent on external services, for example, authentication service on a database. There is also a management Graphical User Interface (GUI) for the smart home user to control the environmental settings. The user must validate against the authentication service before permission is given to the user to control the Smart Home.

The OSGi framework hosts the services for the Smart Space. Packages can be downloaded and deployed on demand and removed when they are no longer needed by a management agent. Deployed bundles can register a number of services that can be shared with other bundles. The OSGi bundle is the only way to deploy services on the OSGi framework. The OSGi Smart Space bundles take the form of a java archive file and are comprised of java classes together with:

- Manifest file, which describes the contents of the JAR file. This contains information such as the location of the Activator Class. The OSGi Framework Specification [15] defines other manifest headers such as Export Package and Bundle Classpath.
- States dependencies on other resources, such as Java packages.
- Bundle Activator, which implements the Activator interface to start a bundle.

IBM's Autonomic Management Engine (AME) is assigned its resources through a process known as initial provisioning [9], Smart Spaces can be assigned a Management Agent with the same process. This Management Agent can then manage the Smart Space environment. As illustrated in Fig 2, a number of bundles were developed and deployed in the OSGi enabled Smart Space.

Deployed Smart Space bundles within the OSGi framework register themselves to the framework so they and their supporting packages can be used by other bundles. In this scenario the environmental control bundle uses the heating and the air condition bundle. The services within the framework are also fitted with service listeners which comply with filters and listen to the registry activities specified by these filters. The purpose of the listeners and filters is to capture service errors within the framework as part of the problem detection process.

The OSGi framework also provides these services with a general purpose logger in the form of a Log Service. The Log Service allows bundles to log information including exceptions, which are crucial to the problem detection element of the autonomic life-cycle. Services log any information they receive via their service listeners which sometimes a faulty service may not get the opportunity to report. The Log Service reports all incoming requests to a local log file. The log entries identify the reporting component, the reported component and the information or error associated with the entry. An example of an error log entry using the environmental scenario may be:

```
ERROR    20060117 10:22:14 bid#44       - origin:EnviromentControl
componement:heatingService issue:unavailable
```

The services now generate state information to be used in problem detection. To avoid having to develop unique management components to consume this information, the standard log format, CBE, is used. CBE can be applied in two ways. Firstly, in the case of an existing application, which can not be easily modified to produce CBEs, an adapter can be put in place to provide the conversion. Conversion can be achieved using the GLA tool provided by IBM [10]. New resource applications should conform to the CBE standard and directly produce CBEs, which can be developed with the use of existing API's. Fig. 2 shows the OSGi application generating CBEs indirectly into the CBE File via the Generic Log Adapter (GLA) engine. The CBE model uses Extensive Markup Language (XML) schemas. The main data components in the CBE specification are data to identify the component logging, data to identify the component that is affected and data about the problem. Using the error example above these are `EnviromentalControl`, `heatingService` and `unavailable` respectively. The CBE model tackles the two big issues of logging standards, which are format and content. The CBE model ensures consistency of format through an XML schema definition.

The next step in architecture development was to produce CBEs from the proprietary log entries. The GLA engine uses our GLA file, to parse proprietary OSGi service log messages to produce CBEs. The GLA file is based on XML and regular expressions [16]. An eclipse plug-in is supplied for the production of this file. A regular expression was created to identify a new entry in the OSGi log file. Regular expressions were also written to parse the log entry into components that apply to the CBE XML schema such as *(.*)origin:validation(.*)*. This example searches for the reporting component in the OSGi log entry. The GLA engine stores the CBE entries, based on Outputter XML property, in a CBE file for further processing as illustrated in Figure two.

CBEs are consumed by the AME and stored by a CBE DataStore for processing in the control loop. The control loop is represented by IBM's Tivoli Resource Model [17]. The Resource Model was built using the resource model builder for CBE consumption, analysis along with solution planning and deployment. Tivoli Resource Models provide a mechanism for supervising and delivering autonomic self-healing properties. The resource model consists of events, thresholds and a VisitTree for parsing any new CBE. The pure JavaScript VisitTree uses CIM classes to monitor its resources and runs based on a preset cycle time.

The resource model receives information on a monitored resource via interfaces such as CIM/WMI. IBM's AME acts as a CIM Manager and provides a platform for CIM classes to run on, along with scripts provided by the resource model. The development of the resource model involved the specification of information that controls the behaviour of the model. Such information includes thresholds the system does not want a resource to exceed, events to generate if the state of a resource is poor, automated corrective actions to specific events and parameters to identify particular resources of interests. In the case of the environmental control scenario above the visit tree was programmed to watch for failings in the heating and air conditioning services. These thresholds and events along with predefined solutions, to the scenario, can be used to manage a smart space. All this information is expressed in a decision tree script.

The IBM Resource Model Builder tool [18] allowed for the creation, modification, testing and packaging of resource models for deployment in the AME. The output of the resource model builder after our development was complete contained the following components:

- Scripts, including the visitTree () function, which contains the monitoring algorithm that is called when needed. Other monitoring functionality required should be added in here as well.
- A Model Object Format (MOF) file, which defines the format of data objects to be monitored. M12 Java providers invoke classes that assist in translating monitored objects into CIM classes. The AME uses CIM-M12 [9] to model resources.
- A Configuration File, which contains platform specific information such as cycle time in seconds.
- A Message catalogue, which contains various parameters such as the model name, and
- A Resource Model Settings file, which contains all the tags and corresponding data elements to execute the resource model.

Later versions of the IBM Toolkit use a Common Manageability Model (CMM), which allows for the asynchronous sending of CBE's and the VisitTree is Java based.

The Resource Model for the OSGi Smart Space was then plugged into IBM's Simple Agent Reference Architecture (SARA), as depicted in Fig. 2, which is a straightforward implementation of IBM's AME. It provides a control interface for loading up new Resource Models and creating instances of them. Once the Resource Model has started it checks for any new CBE, decides on their importance and checks them against predefined corrective actions. Once a solution is found it is deployed via the Java Management Extensions (JMX) server, which resides on the OSGi platform - this is known as the effector. JMX provides a method of managing and monitoring devices and applications on the OSGi framework remotely. In the proposed architecture JMX, which provides a simple method for solution planning and deployment, was used, however future work will explore more complex methods such as Installable Units [13].

In the environmental control scenario, once the environmental service logs one of its services as missing, the log is converted to a CBE and ready for processing by the AME. After identifying the fault component in the VisitTree, a corrective action is found and a batch file is called from within the

VisitTree, with certain arguments to resolve the problem detected. The JMX client is then started with the same arguments and based on these arguments the JMX client connects to the remote JMX server residing on the OSGi platform. Finally the appropriate MBean is invoked that contain interfaces to the known faulty component and applies the necessary changes. Changes could include a restart or a reconfiguration of the live service.

Conclusion

The above architecture demonstrates the overlap between the Autonomic Computing domain and Smart Spaces, using the OSGi framework to provide a more reliable, robust and scaleable environment with some self-healing properties. It is concluded, from the architecture, that the convergence of Autonomic Computing and Pervasive Computing could result in reduction of costs and complexity involved in managing pervasive environments. It can be seen that a generic problem detection solution such as IBM's CBE can be applied to any Smart Space environment. Generic problem detection coupled with autonomic concepts and technologies should deliver a resilient Smart Space of the future.

IBM's Autonomic Computing toolkit is by no means a solution to complexity but it is a step in the right direction. The toolkit contains a flurry of existing technologies and tools that when combined under the goal of autonomics provide a mapping to the requirements and issues of a self-managing Smart Space. The IBM toolkit provides an interesting solution to complexity while delivering self-management in Smart Spaces but the toolkit itself brings its own complexity in development and deployment. High level policy's are a very important part of self-management and are not very well represented in the toolkit for now.

The OSGi framework deals with the current problem of proprietary development in Smart Spaces today. Current Smart Spaces are very closely coupled and complex thus hampering their potential growth. OSGi provides an open environment that accommodates the co-existence of all heterogeneous devices and protocols. OSGi also complies with SOA allowing for composition of services and service lookup. It provides the flexibility and openness that is defined by the SOA concept and also allows for remote management.

5 Future Research

Two essential components are missing from the proposed architecture, a context management component and the knowledge component. The current architecture, does not address the many issues associated with context management and knowledge representation. For this architecture to be a truly autonomic system it must possess some form of knowledge reasoning and context detection. Extensive research is now being carried out to accomplish a higher class of autonomic behaviour. Ontology Based Semantics, is a research area being pursued to achieve dynamic knowledge representation [19]. This research will provide a more in-depth picture of what is needed from knowledge engineering to attain a higher level of autonomics in Autonomic Smart Space environments.

The implementation in this project concentrated on self-healing and therefore further research into the other three Autonomic Computing 'self's' is required to achieve a higher autonomic level. In order to demonstrate fully the concepts the IBM toolkit provides, more complex scenarios would need to be devised. Self-management and all its umbrella terms as mentioned in section 2.1 need to be applied to the Smart Space environment to provide a holistic view of Autonomic Smart Space management.

Solution planning and deployment is another key area in the Autonomic Computing domain that requires further exploration in this existing Smart Space architecture. JMX technology was used to deploy solutions on the Smart Space in the current architecture and does not provide the stable solution deployment architecture described in section 3.2. A more generic scalable solution deployment method needs to be developed. Further research into solutions such as InstallAnyWhere and InstallSheild would greatly enhance this aspect of the architecture.

This research demonstrates how, using IBM's Autonomic toolkit, the concept of self-healing can be applied to a Smart Spaces to provide a more reliable and robust environment. The application of autonomic concepts, such as the three self's, should provide a self-managing Smart Space environment.

References

[1] M. Weiser, "The Computer for the 21st Century," *Scientific American*, vol. 265, pp. 66-75, 1991,http://www.hiit.fi/u/reti/interests/computer-for-21-century.pdf.

[2] R. Murch, *Autonomic Computing*, vol. 1, First ed: Jeffery Pepper, 2004.

[3] J. O. Kephart, "The Vision of Autonomic Computing," in *IEEE Computer Magazine*, vol. 36, 2003, pp. 41–50.

[4] S. Helal, "The Gator Tech Smart House: A Programmable Pervasive Space," *Computer*, vol. 38, pp. 50-60, 2005,http://csdl.computer.org/dl/mags/co/2005/03/r3050.pdf.

[5] Van der Meer S, Davy A, "Ubiquitous Smart Space Management," presented at 1st International Workshop on Managing Ubiquitous Communications and Services (MUCS), 2003.

[6] O'Sullivan, D. and Wade, V, "A smart space management framework," Trinity College, Dublin TCD-CS-2002-23, 2002.

[7] OSGi_Alliance, The OSGi Service Platform - Dynamic services for networked devices, 1999-2005, http://www.osgi.org/, 18/10/2005.

[8] Sun, JavaTM 2 Platform Security Introduced, 1995, http://java.sun.com/j2se/1.3/docs/guide/security/, 8/11/2005.

[9] IBM, "A Practical Guide to the IBM Autonomic Computing Toolkit," vol. 1, 2004.

[10] IBM_ACT, Autonomic Computing Toolkit, 2005, http://www-128.ibm.com/developerworks/autonomic/overview.html?ca=dti-tileautonomic&S_TACT=105AGX01&S_CMP=TILE, 18/10/2005.

[11] IBM_CBEL, Common Based Event Logging, 2005, https://www6.software.ibm.com/developerworks/education/ac-configure/, 18/10/2005.

[12] OASIS, Organization for the Advancement of Structured Information Standards, 1993, http://www.oasis-open.org/home/index.php,

[13] IBM_SDSG, Solution Installation and Deployment Scenario Guide, 2004, ftp://www6.software.ibm.com/software/developer/library/autonomic/books/fqh3mst.pdf,

[14] S. Helal, "Enabling Smart Spaces with OSGi," presented at IEEE ComSoc, 2003.

[15] OSGi_R4, OSGi Service Platform Core Specification The OSGi Alliance, 2005, http://www.osgi.org/osgi_technology/download_specs.asp?section=2, 19/10/05.

[16] G. Jan, Regular-Expressions.info, 2006, http://www.regular-expressions.info/tutorial.html,

[17] Tivoli, IBM Tivoli Resource Model Builder, 2005, http://www-306.ibm.com/software/tivoli/resource-center/availability/code-monitor-resource.jsp,

[18] IBM_TRMB, IBM Tivoli Resource Model Builder, 2005, http://www-306.ibm.com/software/tivoli/resource-center/availability/code-monitor-resource.jsp,

[19] David Lewis, Kevin Carey, Thanassis Tiropanis, Simon and Courtenage, "Semantic-based Policy Engineering for Autonomic
Systems," presented at 1st IFIP WG6.6 International Workshop on Autonomic Communication - Autonomic Communication Principles, 2004.