

# Policy-Based Autonomic Control Service

N. Badr  
*School of Computing and  
Mathematical Science,  
Liverpool John Moores  
University,  
Byrom Street, Liverpool  
L3 3AF, UK*  
[cmsnbadr@livjm.ac.uk](mailto:cmsnbadr@livjm.ac.uk)

A. Taleb-Bendiab  
*School of Computing and  
Mathematical Science,  
Liverpool John Moores  
University,  
Byrom Street, Liverpool  
L3 3AF, UK*  
[a.talebendiab@livjm.ac.uk](mailto:a.talebendiab@livjm.ac.uk)

D. Reilly  
*School of Computing and  
Mathematical Science,  
Liverpool John Moores  
University,  
Byrom Street, Liverpool  
L3 3AF, UK*  
[d.reilly@livjm.ac.uk](mailto:d.reilly@livjm.ac.uk)

## Abstract

*Recently, there has been a considerable interest in policy-based, goal-oriented service management and autonomic computing. Much work is still required to investigate designs and policy models and associate meta-reasoning systems for policy-based autonomic systems. In this paper we outline a proposed autonomic middleware control service used to orchestrate self-healing of distributed applications. Policies are used to adjust the systems autonomy and define self-healing strategies to stabilize/correct a given system in the event of failures.*

## 1. Introduction

Recently, there has been a considerable interest in both policy-based management, goal-based service management and autonomic distributed systems management [1-9]. Policies are often encoded as rules to specify for instance systems security, management properties, allowable states and/or obligations. Based on an ongoing study into requirements for self-adaptive software engineering and management, this paper outlines the interplay of self-managing and self-healing policies and deliberative reasoning models to orchestrate, enable and/or adjust application autonomy. The paper describes a prototype policy-based autonomic system, which is deployed as a middleware service. The system is based on: *service manager*, *system controller* and *JavaSpaces*, and self-healing policy repository.

The remainder of the paper is structured as follows: section 2 provides background material relating to our approach and in particular describes current trends in autonomic-based management and policy-based management approaches. Section 3, provides an overview of the policy-based autonomic control service. Section 4,

describes the development of the control service based on Jini middleware technology. Section 5, presents a case study and section 6, draws overall conclusions and mentions future work.

## 2. Background

Recent research has focused on the development of methods and techniques to support runtime management and self-governance of distributed applications, which is reviewed briefly below.

### 2.1 Policy-Based Management

Much research has addressed the use of policies for dynamic management of large distributed systems [1-3]. Moffett *et al.* [3] stated the necessity of representing and manipulating policy management of distributed systems. In policy management, action policies are represented as a policy hierarchy, where each policy in the hierarchy represents a plan that meets a specific objective. Sloman and Lupu [10] studied authorization and obligation policies' specification for programmable networks. Such policies are interpreted to facilitate runtime activation, deactivation and/or their modification without having to shutdown the network node. Other approaches to policy-based management have used condition-action rules to support a static policy configuration-based solution, in which human intervention is required for system reconfiguration and policy deployment. Moffett *et al.* [11] have proposed a framework for supporting automated policy deployment and flexible event triggers to permit dynamic policy configuration, focusing on solutions for dynamic adaptation of policies in response to changes within the managed environment. Other research efforts have focused on policy specification and enforcement for dynamic service management. For instance, the IETF

Policy Working Group is developing a QoS network management framework using the X.500 directory service, where policies are encoded as If-Then rules and stored in directories [12].

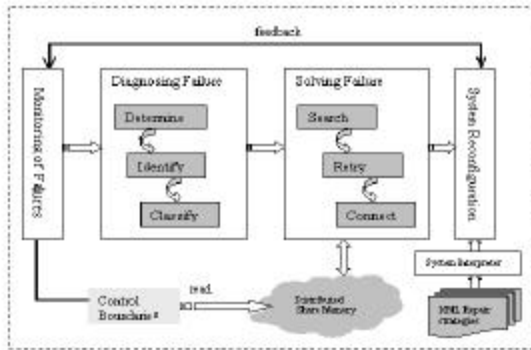


Figure 1: Autonomic management process model.

## 2.2 Autonomic-Based Management

Within the distributed system community work is underway to understand and design autonomic computing systems endowed with self-management and abilities to adjust to unpredictable changes [13]. The two main elements of autonomic management are the *functional unit* and the *management unit*. The functional unit performs the main operation and is provided by elements such as web services or databases. The management unit is responsible for system resources and operational performance and hence the reconfiguration of resources according to adaptive changes [5]. Autonomic systems have been defined by IBM [6] as system that have “... *The ability to manage themselves and dynamically adapt to change in accordance with policies and objectives...*”. In other words, they have the ability to monitor, diagnose and heal themselves. This entails that systems have the ability to dynamically insert and remove code at runtime. *Hot swapping* [8, 9] is proposed as a means to enable autonomic software systems to *interpose and/or replace* components (code) in response of either failure or software maintenance.

## 3. Policy-Based Autonomic Control Service

Autonomic systems must discharge the intended self-management functionality in a safe, controllable and predictable manner avoiding any emerging, accidental errors or undesirable features [9-12]. This therefore requires that autonomic systems extend and integrate autonomic control models with policy-based service [9]. This formed the motivation for our work, which provides a self-governance mechanism based on a developed

Extensible Believe, Desire and Intension (EBDI) model for deliberative systems. The EBDI model is used to guide and manage an autonomic application’s self-healing processes.

Figure 1 illustrates the autonomic management and interaction model used to monitor an application. The overall process starts with the monitoring process to detect conflicts using the control service internal and external policies. This followed by the diagnosis process and then the repair strategies, which are based on the proposed EBDI model (Sec. 4). The strategies are embedded in an external format such as XML, which may be parsed and translated into executable format by the system interpreter. In addition the distributed shared space is used to facilitate the remote integration and coordination of the distributed control services.

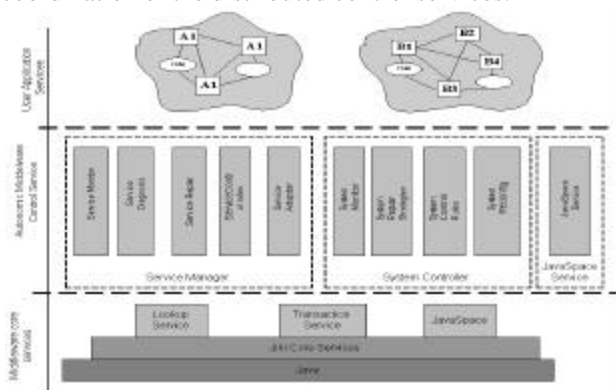


Figure 2: The control service architectural layers.

## 4. Design of the Proposed Control Service

As shown in Figure 2, the design of the policy-based autonomic control service combines three main services:

- *Service Manager*: is used to adapt the structural components and the dynamic behaviour of the services they provide. Structural components can evaluate their behaviour and environment against their specified goals with capabilities to revise their structure and behaviour accordingly [20]. This separation into distinct service managers eases their management by decentralizing the control of each individual application-level service. Service managers look after their application-level service and monitor its behaviour using an external and internal policies.
- *JavaSpace Service*: is a persistent distributed shared memory used by the self-healing process for awareness and coordination. It stores the required information and reports service states for use by the system controller service to repair and reconfigure the system in the event of any

service failures. The JavaSpace service uses *external policies* to notify the system controller directly with remote events. The remote events may either notify of changes in application-level service states (as reported by the service manager) or notify the system controller if the lease of a service manager should expire.

- *System Controller*: is responsible for the dynamic reconfiguration of a given application by coordinating the activities of individual service managers. Each manager is used as a meta-service to monitor, repair and adapt its associated application-level services in the event of conflicts/failures. Service managers may then report their associated application-level service states using the JavaSpace service. Associated external policies can either trigger a notification event to the system controller or request a system controller service's activation to monitor a given application service states. For example, the action shown below may be used if the average latency of the control process exceeds the maximum allowed latency. If such a case occurs the control rule detects and triggers an execution failure notification leading to a self-healing process.

```

If (control_avLatency is larger than
    control_maxLatency)
start
    conflict_monitor(true);
    start_control_process();
end;

```

The development of system repair strategies and self-healing control is based on our proposed Extensible Beliefs, Desires and Intension (EBDI) model [11-13]. The model is concerned with situated intentional software that continuously monitors and/or observes its environment and acts to changes in accordance with its sets of policies.

This architecture is based on a control service model that follows a cycle of monitoring the target application, detecting undesirable behaviours (events), identifying conflicts/errors, prescribing remedial action plans and enacting change plans through reconfiguration<sup>1</sup>.

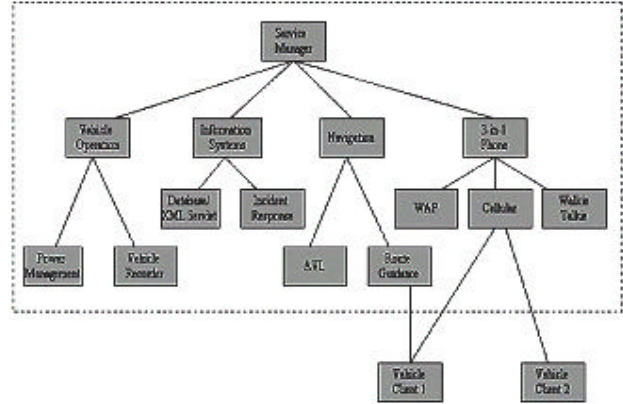


Figure 3: The Architectural view of the EmergeITS application.

## 5. Case Study

The policy-based autonomic control service has been developed and demonstrated through an industrial case-study, namely EmergeITS [12]. The latter was developed using Jini middleware as a proof-of-concept for self-healing software for intelligent networked appliances (vehicles). The case-study was developed in collaboration with the Merseyside Emergency Fire Service (Fig. 4). One of the EmergeITS services evaluated was the 3-in-1 phone, which allows a mobile phone or PDA device to be used in one of three different modes: a cellular phone, a WAP phone or a walkie-talkie. The 3-in-1 phone may be used for either voice communication or to receive multimedia content, subject to the requirements of the user and availability of a communication service provider. An autonomic middleware control service prototype was incorporated into the case study to provide the meta-control software to support EmergeITS over the network.

The 3-in-1phone service is hosted by an in-vehicle computer, which also acts as a gateway. If the local service deployment should fail, then a service manager is started to provide self-monitoring, self-diagnosis, self-repair and self-adaptation. In the event of failure, the service manager will select an appropriate repair strategy, thereby providing a degree of conflict resolution and fault tolerance. The service manager stores the 3-in-1 phone service state in a JavaSpace service and the system controller may check these states against external policies.

<sup>1</sup> A full description of the design and implementation of the autonomic middleware control service is out of the scope of this paper and can be found in a longer version of this paper [12].

```

<?xml version="1.0" ?>
-<!-- Simple Description of 3in1 phone Strategies -->
-<!--
-<DOCTYPE strategy [View Source for full doctype...>
-<Strategies>
-<Strategy id="1" type="desire">
-<Action id="1" type="plan" name="Connect">
-<Properties>
  <property id="1" name="host"<cmrnbadr/>property>
  <property id="2" name="Location">GPS_Loc</property>
  <property id="3" name="Max_connected">maxNo</property>
  <property id="4" name="Method">connect</property>
</Properties>
</Action>
</Strategy>
-<Strategy id="2" type="intention">
-<Action id="1" type="plan" name="Retry">
-<Properties>
  <property id="1" name="file_name">file</property>
  <property id="2" name="ServiceStatus">not null</property>
  <property id="3" name="Method">connect</property>
</Properties>
</Action>
-<Action id="2" type="plan" name="Alternative">
-<Properties>
  <property id="1" name="host"<cmrnbadr/>property>
  <property id="2" name="New Manager Interface">ManagerProxy</property>
  <property id="3" name="Get Manager">getServiceManager</property>
  <property id="4" name="Client Interface">ClientProxy</property>
  <property id="5" name="Get Client">getClient</property>
  <property id="6" name="Notify Client">notifyClient</property>
  <property id="8" name="Max_connected">maxNo</property>
  <property id="9" name="Method">connect</property>
</Properties>
</Action>
</Strategy>
</Strategies>

```

Figure 6: The XML description of a self-repair strategy.

## 7. Conclusions and Future Work

In this paper we have described the development of a policy-based autonomic control service, which has been implemented using Jini middleware technology. The control service incorporates the policies (either internal or external) within the architecture of the autonomic control service to achieve policy-based autonomic control of distributed applications at run time.

We have briefly described the proposed Extensible BDI (EBDI), which is a policy (normative)-based model that is used to examine the system external policies and for generating the appropriate repair strategy at runtime.

We have described the main services of our policy-based autonomic control service (service manager, JavaSpace service and system controller service) and demonstrated their use to control a 3-in-1-phone application service. In addition we have evaluated the performance of the autonomic control service using the elapsed time and the average latency as metrics of the system at runtime.

In our future work, we intend to improve/enhance the autonomic control service on two main fronts: self-protection and machine learning. There is a need to provide self-protection to provide security in untrustworthy environments. There is also a need to equip the current model with the machine learning capabilities so that the policy-based autonomic control service can access history and knowledge relating to the previous failure cases.

## 8. References

1. E. Lupu, et al. *A Policy Based Role Framework for Access Control*. in *First ACM/NIST Workshop on*

*Role-Based Access Control*. 1995. Maryland USA: ACM.

2. J. Moffett and M.Sloman, *Policy Hierarchies for Distributed Systems Management*. IEEE Journal on Selected Areas in Communications, 1993. **11**: p. 1404-1414.
3. K. Yoshihara, M.Isomura, and H.Horiuchi. *Distributed Policy-based Management Enabling Policy Adaptation on Monitoring using Active Network Technology*. in *12th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*. 2001. Nancy France.
4. D. Chess, C.Palmer, and S.White, *Security an autonomic computing environment*. IBM SYSTEMS JOURNAL, 2003. **42**.
5. J. Appavoo, et al., *Enabling autonomic behavior in systems software with hot swapping*. IBM SYSTEMS JOURNAL, 2003. **42**.
6. M. Sloman and E. Lupu. *Policy Specification for Programmable Networks*. in *First International Working Conference on Active Networks (IWAN'99)*. 1999. Berlin.
7. K. Barber, et al. *Conflict Representation and Classification in a Domain Independent Conflict Management Framework*. in *the Third International Conference on Autonomous Agents*. 1999. Seattle WA.
8. P. Horn, *Autonomic Computing: IBM's Perspective on the State of Information Technology*. IBM Corporation, 2001.
9. D. Reilly, et al. *An Instrumentation and Control-Based Approach for Distributed Application Management and Adaptation*. in *Workshop on Self-Healing Systems (WOSS'02)*. 2002. Charleston SC USA.
10. N. Badr, D. Reilly, and A. Taleb-Bendiab. *A Conflict Resolution Control Architecture for Self-Adaptive Software*, in *the International Workshop on Architecting Dependable Systems: WADS 2002 (ICSE 2002)*. 2002. Florida USA.
11. N. Badr, *An Investigation into Autonomic Middleware Control Service to Support Distributed Self-Adaptive Software*, PhD thesis, School of Computing and Mathematical Sciences, Liverpool JMU, 2003, www.cms.livjm.ac.uk/cmsnbadr.
12. N. Badr, D. Reilly and A. Taleb-Bendiab, "Policy-Based Autonomic Control Service", Technical Report, School of Computing and Mathematical Sciences, Liverpool JMU, 2004, www.cms.livjm.ac.uk/cmsnbadr.
13. M. Bratman, *Intentions, Plans, and Practical Reason*. Harvard University Press, 1987.