A Conflict Resolution Control Architecture For Self-Adaptive Software

N. Badr
School of Computing and Mathematical Science,
Liverpool John Moores University,
Byrom Street, Liverpool
L3 3AF, UK
cmsnbadr@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

A.TalebBendiab
School of Computing and Mathematical Science,
Liverpool John Moores University,
Byrom Street, Liverpool
L3 3AF, UK
A.Talebbendiab@livjm.ac.uk

ABSTRACT

An essential feature of dependable software is its adaptive capability to respond to changes that occur in its operating environment through the dynamic transformation and reconfiguration of its components and/or services. Such adaptive capability is often a design aspect derived from the software architecture model, which describes the software components and their interactions, the properties and policies that regulate the composition of the components and norms that limit the allowable systems adaptation operations. Research in reflective middleware architectures and policy-based distributed systems management has focused on the use of managerial or meta-level protocols to attain reactive adaptive behaviour. However, reflective and policy-based management approaches alone cannot address all of the needs of self-adaptive software due to their inability to maintain a faithful runtime model of the system. This paper considers the development of control architecture for self-adaptive software, which combines conflict resolution and control strategies to resolve runtime conflicts. In particular, the paper describes a prototype service-based architecture, implemented using Java and Jini technologies, which provides runtime monitoring and conflict resolution to support software self-adaptation.

.

1.Introduction

Self-adaptive software can be seen as a new architecture style, which extends the controller concepts to adapt the structural configuration and dynamic behaviour of a system.

Structural components can evaluate their behaviour and environment against their specified goals with capabilities to revise their structure and behaviour accordingly. *Laddaga* [7] defines self-adaptive software as:

"*Software that evaluates and changes its own behaviour when the evaluation indicates that it has not accomplishing what it is intended to do, or when better functionality or performance is possible*".

Such a software architecture style presents an attractive concept to developing self-governing software, which fully or partially accommodates its own management and adaptation activities. Research in this area has adopted control engineering concepts, as typified by *Osterweil* [9] who presents an architecture, which uses a controller with a well-specified control function with *feedforward* and *feedback* loops to enable a target system to be monitored to regulate its operation in accordance with its given control model. *Osterweil* [9] describes the delegation of the responsibility for testing and evaluation of software applications from humans onto automated tools and processes, advocating the automation of the continuous self-evaluation processes.

However, there are further issues to be addressed in order to achieve self-adaptation, such as: reasoning, control and decision-making to assess the gap between a given software operational model and its requirements, and the use of appropriate strategies for conflict resolution. This paper argues, that during any software self-adaptation process, it is likely that autonomous changes may lead to execution errors and software integrity conflicts. Thus the self-adaptation of distributed software requires control and decision-making to support the monitoring, detection and resolution of conflicts, which may occur at runtime.

The remainder of the paper provides an overview of our "work in progress" concerning the development of a service-based architecture that uses conflict resolution to achieve self-adaptation. The paper is structured as follows: section 2 provides a brief review of self-adaptive software and conflict

resolution strategies. Section 3 describes the prototype conflict resolution control architecture and its constituent services. Section 4 briefly describes a case study, which illustrates how the architecture achieves runtime self-adaptation. Finally, Section 5 draws conclusions and mentions future work.

2. Background

The control theory based paradigm provides a framework for designing software that supports self-control during the operation of the software. The self-controlling software model supports three levels of control: feedback, adaptation, and reconfiguration [6]. *Meng* [8] proposed a control system for self-adaptive software based on a descriptive model of a self-adaptive control system, which employs the control system concepts of feedforward and feedback. For example, if a self-adaptive software system consists of two components the feedforward process can provide specifications of the software and its predictability and the feedback process can gather and measure the software's environmental attributes.

Central to this paradigm are the decision-making and delegation strategies that are used to resolve conflicts, as considered by *Barber et al* [2] who discuss different decision-making strategies required in conflict resolution. *Barber et al* describe *negotiation* as the most popular strategy, but also consider *arbitration*, *mediation* and/or *voting* as viable strategies in agent-based systems. *Adler et al.* [1] describe the "Independence" strategy, which regards *self-modification* as a simple and effective resolution strategy for use in agent-based systems, which is used when an agent detects a conflict but does not wish to interact with other agents to solve the conflict, as the agent would rather resolve the conflict itself. Other approaches by *Williams and Taleb-Bendiab* [13] illustrate the use of meta-languages to support software agent composition and the runtime reconfiguration of middleware services.

Recently there has been an increasing research trend in the development of *self-healing* software, facilitated through innovations in operating systems [5] and *reflective middleware* architectures through which structural and/or behavioural models can be modified at runtime. *Robertson et al* [12] indicate that whilst reflective architectures share similar aims with self-adaptation architectures, they differ in that self-adaptive architectures generate runtime evaluators to check the deviation of the state of the program against some measure. A control regime is then used to compute the distance between the current state and the goal state in order to maintain stability and robustness. The control regime makes use of sensor and actuator concepts to feedforward and feedback the systems states enabling the software management layer to reconfigure and switch the control regime itself to suite the systems requirements.

The general model of self-adaptive software can be viewed from many different aspects, which take the architectural model as a parameter in the monitoring and repairing framework to allow the monitoring mechanism to match both properties of interest and adaptation operators at runtime [3]. Previous work by *Cooper and Taleb-Bendiab* [4], based on the same theme as the current research described herein, focused on a heuristic-based approach to support software agent self-adaptation. The current research extends this previous work by concentrating on automated self-adaptation that can be applied at runtime.

3. Service-Oriented Conflict Resolution Control Architecture

The development of dependable distributed system is hindered by the conflicts and faults that may occur at runtime. With being the case, our approach is based on a control mechanism which monitors behaviour, detects and identifies conflicts and formulates remedial action in the form of a resolution strategy. A *service-oriented* approach was adopted to develop the control mechanism and overall service-based architecture, as shown in Figure 1 overleaf.

The service-based architecture achieves self-adaptation by detecting, identifying, and resolving conflicts that occur at runtime, After detection, a conflict is identified and categorized according to its type before a resolution strategy is used to *minimize* the conflict. A monitor element then provides feedback to guide the conflict resolution tasks, which are used to implement the conflict resolution process. The prototype architecture, used to implement the conflict resolution and control element (figure 1) is based on Java and Jini middleware[1] technologies to provide the following services.

❖ Monitor: makes use of a set of control rules against which behaviour is monitored to detect conflicts.

❖ Diagnostic: the execution of a control rule implies a conflict, which activates the diagnosis services that results in:

- Identification of the part of the control rule that raised the conflict.
- Identification of the cause of the conflict through the examination of service attributes and method invocations using Java's reflection API (`java.lang.reflect`).
- Classification of the type of conflict, which provides the basis for the selection of a resolution solution strategy.

❖ Notification: makes use of Jini's remote event mechanism to notify clients when conflict resolution solutions become available.

❖ Control Rules: serve as the basis for the previous monitoring and diagnose services. They consist of a number of rules and gates that execute when a conflict is detected, which in practice execute when a service method is invoked. This in turn may result in the firing of a remote event to notify of the availability of a resolution solution strategy.

❖ Exception Handling: makes use of Java's exception handling facilities to catch exceptions, which are thrown when a control rule executes due to some kind of fault/attack that cannot be solved. Exceptions are dealt with according to priority, which may be low, intermediate or high to accommodate varying degrees of fault tolerance.

[1] Jini is a Java based middleware technology developed by Sun Microsystems Inc.<http://www.sun.com/jini/specs>

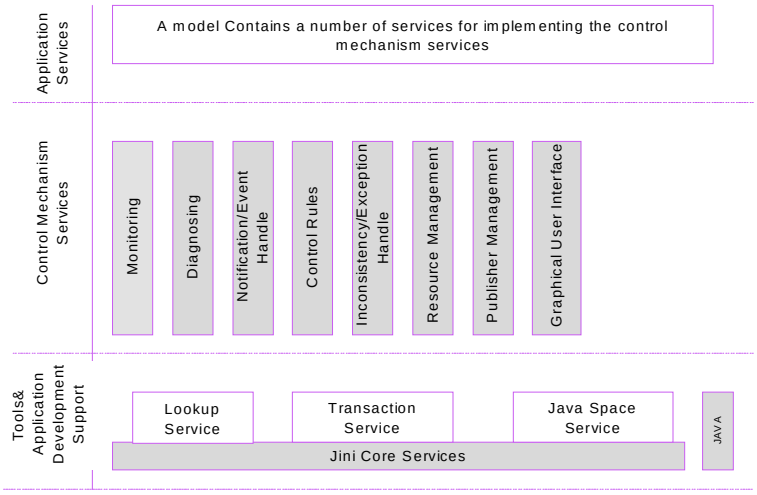

Figure 1: Self-Adaptive Control Architecture

Figure 2 illustrates a flow chart of the reasoning that takes place in a typical conflict resolution solution strategy. The flowchart begins with a client request for an application service and if the request succeeds, then no conflict has occurred so the client responds. If a conflict does occur an attempt is made to provide the client with an alternative service, which performs the same function. If this also results in a conflict then the client may choose a renewable notify option, whereby the client is notified when a suitable service becomes available or the client may renew the notification period when its lease expired.

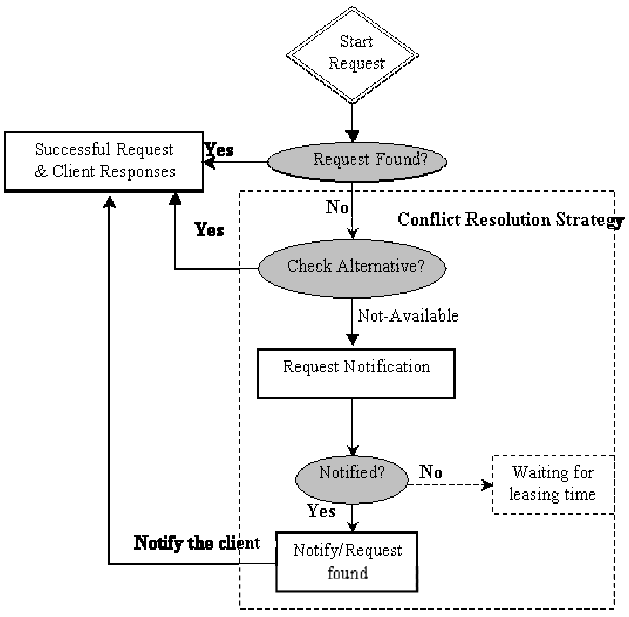

Figure 2: Example of Control Resolution Strategy

4. Case Study

We illustrate the architectures capabilities to accommodate self-adaptation through an industrial case-study, which involves a dependable software system developed out of the EmergeITS project. EmergeITS is concerned with the development of an adaptable software architecture to provide In-Vehicle Telematics Systems (IVTS) capabilities to emergency service response teams [11]. Figure 3, provides an overview of the EmergeITS services, which use Jini services to provide: remote hardware control, remote database access (through Java Servlets and XML documents) and mobile communication management capabilities. The IVTS Manager oversees the overall operation of the system and provides capabilities to add additional services, as well as limited control and adaptation facilities. IVTS Clients (typically remote vehicles) may request the use of any service that the IVTS Manager has to offer.

One crucial service for the IVTS architecture is the *3-in-1 phone* service (*palowireless2001*) [10], which allows a portable wireless phone or PDA to be used as a cellular phone, WAP device or walkie-talkie. An IVTS Client may request the use of a 3-in-1-phone service whenever a physical device such as a mobile phone or PDA is to be used from a moving or stationary vehicle. The client may request to use the device in one of the 3 different modes for which the 3-in-1 phone service must adapt accordingly. The 3-in-1 phone service essentially implements the conflict resolution and control strategies by examining: 1) user operation mode requested, 2) client location and 3) bearer service location and availability (e.g. BTCellNet, , GSM or Tetra[3]) and each of these three parameters manifest as remote method invocations made on the 3-in-1 phone service. The outcome of the initial resolution strategy may result in IVTS client notification that the request was successful, or else that a further conflict occurred, leading to further control rules

[2] EmergeITS is a collaborative project involving our own research group within the School of Computing and Mathematical Sciences at Liverpool John Moores University and the Merseyside Fire Service <http://www.cms.livjm.ac.uk/emereits>

[3] Tetra is a digital network under consideration by Police and Fire emergency services in the UK

executing for which a conflict resolution must be sought, such as the initiation of a more comprehensive search for a bearer service to resolve the conflict. Eventually a situation is reached whereby the system has attained a configuration that best meets the needs of the initial request.

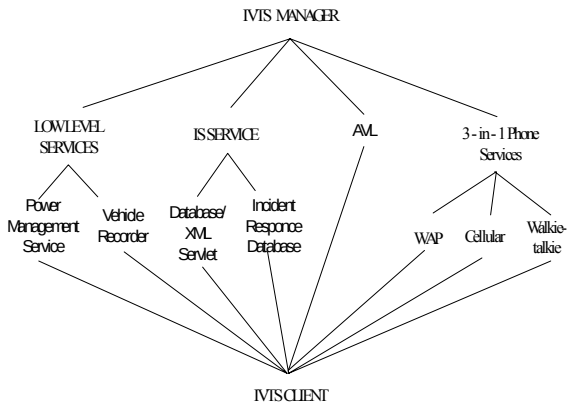

Figure 3: IVTS Services

5. Conclusion and Future Work

In this paper we have described a prototype service-based architecture, based on Java and Jini technologies, that uses conflict resolution and control strategies to detect, identify and resolve conflicts that occur at runtime. We intend to extend our architecture, in future related research, by considering the negotiation aspects in the control decision making of self-adaptation. It is anticipated that this will increase the flexibility of our architecture, which we intend to evaluate through further case studies from our complementary group research area of intelligent-networked-vehicles

References

[1] Adler, M., *et al. Conflict- Resolution Strategies for Nonhierarchical Distributed Agents*. in *In Distributed Artificial Intelligent ||,*. 1989. London.

[2] Barber, K.S., T.H.liu, and D.C.Han. *Strategic Decision-Making for Conflict resolution in Dynamic Organized Multi-Agent Systems*. in *GDN 2000 PROGRAM*. 2000.

[3] Cheng, S.W., *et al.*, *Using Architectural Style as a Basis for System Self-repair*, . 2002.

[4] Cooper, S. and A.Taleb-bendiab. *A High Level ControlMechanism For Managing Conflict Resolution In Concurrent Product Design*. *In Proceedings of the fourth ISPE International conference on Concurrent Engineering :Research and Application (CE97)*. 1997

[5] eLizaProject, http://www-1.ibm.com/servers/introducing/eLiza .

[6] Kokar, M., K. Baslawski, and Y. Eracar, *Control Theory-Based Foundation of Self- Controlling Software.* IEEE Intelligent Systems, 1999: p. 37-45.

[7] Laddaga R. *Active Software*. in *First International Workshop on Self-Adaptive Software, (IWSAS2000)*. 2000.

[8] Meng, A.C. *On Evaluation Self-Adaptive Software*. in *First International Workshop on Self-Adaptive Software, (IWSAS2000),* ***April 2000.*** 2000.

[9] Osterweil, L.J. and Clarke, L.A. *Continuous Self-Evaluation for the Self-Improvement of Software*. in *" First International Workshop on Self-Adaptive Software, (IWSAS2000)*. 2000.

[10] palowireless:Bluetooth Resource Center

[11] Reilly, D.and A. Taleb-Bendaib, *A Service Based architecture for in-Vehicle Telematics Systems* Submitted in "A Special Issue of CERA Journal: A. Complex Systems Perspective on Concurrent Engineering", 2002

[12] Robertson, P., Laddaga, R., and Shrobe H.. *Introduction: the First International Workshop On Self-Adaptive Software*. in *First International Workshop on Self-Adaptive Software, (IWSAS2000)*. 2000.

[13] Williams, M. and A.Taleb-Bendiab. *A Toolset for Architecture Independent, Reconfigurable Multi-Agent systems*. in *First International Workshop on Mobile Agents*. 1998.