

An Knowledge Model for Self-regenerative Service Activations Adaptation across Standards

Mengjie Yu¹, David Llewellyn Jones¹, A. Taleb-Bendiab¹,

¹ School of Computing and Mathematical Science, Liverpool John Moores University,
Byrom St. Liverpool, L3 3AF, UK
{cmsmyu,d.llewellyn-jones,a.talebbendiab}@livjm.ac.uk

Abstract. One of the greatest challenges for dependable service-oriented software systems of next generation is coping with the complexity of required adaptation or reaction to the detected unforeseen vulnerability attacks. To this end, autonomic system[1] has been advocated as a way to design self-protective systems to defend against malicious attacks or cascading failures. However, other initiatives such as the self-regenerative system[2] adopt the biological-inspired [2, 3] notions such as natural diversity and self-immune as a main strategy to achieve the robust and adaptable self-protection. Based on an ongoing research into self-regenerative programming model, this paper presents a knowledge-centric approach for supporting the runtime automated generation of software adapters for cross-standard service activation; and argues the importance of application of a semantic knowledge to extract the notion of self-regenerative adaptation from the previous polyarchical middleware implementation. The benefit of this will be the production of a customizable self-regenerative adaptation service; and also, support for abstraction integration between domain of similar interests or others in a high-level management directed towards building autonomic systems in a large domain of interest.

1 Introduction

“Self” systems, inspired by the concept of autonomic computing introduced by IBM[1], has fuelled a growing interest in the construction of next generation distributed systems, which can efficiently self-manage themselves during the runtime processes without significant users’ interference. Self-healing, as a specific system reconfiguration of autonomic computing, denotes distributed systems abilities to observe failures resulting from faulty software and hardware, and consequently apply appropriate corrections to recover themselves from those unexpected failure. Most of these marvelous high-level decisions or behaviours require huge amounts of complex interaction and integration at the service component level. Moreover, research challenges raised from next generation software development architectures have also desires self-healing systems to build within service-oriented grid computing[3] environment. Components of distributed systems may deploy or implement with various distinct

service standards and middleware technologies. Thus, robust self-adaptation becomes a vital capability for systems to recover themselves from failures by adopting any available grid resource. In these days, numbers of research projects have already addressed this complex service activations adaptation problem. Much insight into use of common interoperation models has been adopted to accommodate adaptation among the diversity of grid components including: OPENWING[4], Model Oriented Architecture[5], and ICENI[6]. These approaches enable systems to interact with resources independently of direct knowledge of one another through a shared common interaction framework or a communication protocol layer. This work has shifted the burden of making interaction and adaptation consistent onto given technologies (e.g. static APIs) and data format (e.g. neutral meta-data such as Web service Description Language-WSDL). In order to apply these standard technologies, each component has to modify their original implementation if they intend to be integrated with or accessed (e.g. discovery and invocation) by other systems or components implemented with different mechanisms. Such approach is viable if suitable configuration and implementation has taken place during design time ahead of system deployment. However, rapidly changing technologies and unpredictable execution environments motivates a desire for more flexible runtime solutions allowing service adaptation across standards. In which case, systems don't need to halt processes for reconfiguring their adaptation setting in order to adopt available grid services bound with different approaches. Such runtime aspect is especially vital for self-immune systems to correct their security vulnerabilities whenever an unforeseen security attack occurs. As a result, numerous diverse versions of security systems can be generated by adopting various security components (such as access controls, encryption data formats, firewalls). Consequently, this approach avoids the generation of new vulnerability that might result from critical security components being halted during reconfiguration.

To solve the problems outlined above, this paper focuses on an alternate research solution: a runtime self-regenerative adaptation model based on a developed polyarchical middleware approach. A detailed description of this proposed polyarchical middleware and its self-regenerative model have been discussed in previous work[7] on supporting multi-standard service interoperation protocols and the adaptation abstract programming model[8]. However, this paper primarily focuses on the application of a knowledge model to extract the notions of self-regenerative adaptation activities from adaptation implementation. This knowledge can extend the polyarchical middleware to support software factory aspects in order to produce various adaptation services tailored to service consumers' own needs. It also supports the sharing of understanding of the self-regenerative adaptation between domains of same interests or others in a high-level management and configuration.

The rest of this paper is organized as follows: it begin with a short overview of the developed self-regenerative middleware service. It is followed by a discussion covering the reasons for use of a knowledge model. This is illustrated using a ongoing design and implementation of the knowledge model. Finally, the paper concludes with a discussion of the future development opportunities it affords.

2 Overview of Self-Regenerative Adaptation Approach

To solve the problems outlined above, this paper will focus on a novel solution: a runtime self-regenerative model for service activation adaptation, based on a developed polyarchical middleware approach[7] that have been developed and implemented in Java. Considering that autonomic systems can spontaneously interact with destination services because they all share the same interaction framework and communication layer, it can be supposed that in certain instances the only suitable service component arriving into the environment will be unexpectedly bound with different implementations. In this case, it is desirable that autonomic systems should still self-configure themselves by adopting the new component. As components have no prior knowledge of one another, a methodology is required to enable these discrete components to communicate with each other. The authors introduce a novel middleware framework that supports autonomic systems by dynamically generating the appropriate adaptation code (i.e. adapters) for the given or available services of choice. With such adapters, systems can discover and interact with remote components without revising individual components to implement special APIs at design time. The authors believe such an approach can well support autonomic systems to continuously maintain and adjust their operation through a high level of self-management while still achieving an alternative choice to interact with various components in unpredictable changing environments.

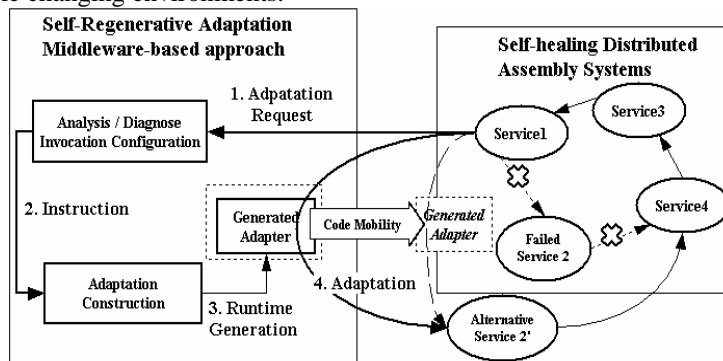


Fig. 1. Self-Regenerative Service Adaptation Model

As shown in Figure 1 taken from [7], we considered a distributed service assembly (application), and at runtime we assumed that *service 2* fails. Suppose *service 2'* is the only available service matching the requested service (*service 2*). However, it implements a different interface (invocation mechanism) not supported by *service 1*. Thus, *adapter* between *service 1* and *service 2'* needs to be auto-generated and deployed. This will be achieved and supported by our polyarchical middleware architecture. In order for this to be achieved, a generic structural model[8] is designed to support our proposed self-regenerative adaptation service to accommodate dynamic loading and the binding of required middleware components. Accordingly the diversity of various service standards and middleware technologies can be achieved through the binding of those on-demand adapters.

3 Knowledge for Self-regenerative Adaptation

The previous section briefs the developed self-regenerative approach for supporting service adaptation across standards. The use of abstraction models[8] provide the polyarchical middleware a generic structure to accommodate the diversity of service standards and middleware technologies adopted for the generation of on-demand adapters. To extend this further, our interest is in developing a semantic knowledge that can extract details of adaptation generation from the polyarchical middleware implementation. This knowledge can be used to support the design of *Service Adaptation Manager* in form of software factory.

3.1 Self-regenerative Adaptation Knowledge

Knowledge development can usually be viewed from two perspectives: 1) “Knowledge=Object” which specifies the semantic interpretation and implicit meanings of various terms and concepts across domains; 2) “Knowledge=Process” which primarily focuses on the service processes composition or software productions improvements. Here, the self-regenerative adaptation knowledge has been designed to describe processes involved with adaptation generation.

3.2 Reasons for Applying Adaptation Knowledge

In brief, the reasons for applying knowledge to support the self-regenerative adaptation service can be summarized as the following terms:-

Separation domain knowledge from implementation is one of the more common goals in developing knowledge. In this case, details of static adaptation procedures will be extracted from the implementation and figured into this domain knowledge. Thus, *Service Adaptation Manager* can be mostly considered primarily as a configuration environment. It will follow process composition models provided by this domain knowledge to produce various adaptation services.

Reuse of domain knowledge is another driving force behind this approach. Domain with the same interest can simply adopt this knowledge by applying specific implementations. Additionally, it considers the building of a large domain knowledge, this existing knowledge can also be integrated and share portions of the large domain description. For example, considering more specific notions or concepts, this knowledge can be extended to integrate with domain interests of self-immune systems (like project PUCSec[9]). In this case after the dynamic evaluation of a system’s security through composition analysis, the security level may be automatically increased through the adoption of components providing various security algorithms or implementations. This may occur as a result of dynamic changes to the system, or whenever a security attack to the system is detected. The reuse of domain knowledge in this way helps to improve the effectiveness and robustness of the overall system.

Process guidance and resource specification can support the design of Service Adaptation Manager into a software factory model, which considers the production of adaptation service families rather than a single approach. This software factory is guided by this knowledge to assemble the on-demand middleware resources for facilitating the adapters construction using the given process models either using default or customized settings.

Knowledge integration plays an important role in the support of high-level information management or integration with domain of the same interests or others. The usages of this term can be viewed from two perspectives: 1) Enables service consumers in other domains to integrate this self-regenerative adaptation service into their system processes. This knowledge can provide sufficient information about “what the self-regenerative adaptation does” and “how it works”. Thus, service consumers can customize adaptation services tailored to their own needs by defining individual adaptation process models. For example, this knowledge can be used in domains with self-healing interests to support distributed systems recovering from the component failures; 2) Improve adaptation knowledge by integrating with domains of same interests. In consideration of a large domain, various adaptation domains can extend their knowledge or share adaptation functions by integrating this high-level representation with each other. This can greatly leverage the existing adaptation resources in considering the large investment on the adaptation applications development.

3.3 Structure of Adaptation Knowledge

To complete the efforts mentioned in previous section, this adaptation knowledge has been process has been designed into a two-layer model as shown in the Fig2: 1) adaptation ontology; 2) middleware components description. Ontology[10], the term borrowed from philosophy, has widely been adopted to describe domain interests in various fields, and also for heterogeneous information integration[11]. The adaptation ontology here describes the self-regenerative adaptation service in term of a process model of the form “what adaptation functions it offers” and “how they work”. This information acts as a task list for software factory- service adaptation manager- on how to produce adaptation generation service, and also provides verification and justification for the adaptation processes. In order to do this, this ontology can further be divided into two layers: adaptation concepts and adaptation process models. The concept layer provides a high-level description for individual adaptation functionalities (e.g. processes) supported by polyarchical middleware. The details of the concept layer consist of various information including: processes description, control structure and data flow structure of processes in terms of the inputs, outputs, preconditions and their sub-processes if applies. The process model layer provides the details of how these processes can actually be composed to facilitate the generation of on-demand adapters. The separation of the adaptation ontology into two sub-layers leaves the configuration of adaptation process models more customizable. As a result, adaptation service consumer can compose process models tailored to their own needs

based on the information from the concepts layer if they don't want to stick with default process models provided by the polyarchical middleware. In addition, information from this adaptation ontology can also support the knowledge integration with domains of same interests or others in a high-level management or configuration.

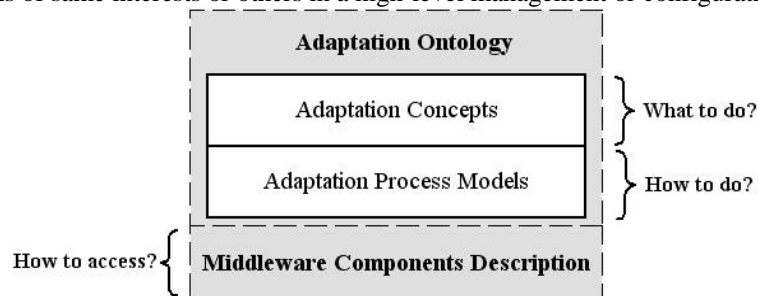


Fig. 2. Adaptation Knowledge Structure

The middleware components description declares the mappings from middleware components with specific implementation to processes declared in the adaptation ontology section. It also describes the accessing details for each middleware component into the technical terms including: protocol, message formats, serialization and addressing etc. In a quick summary, the software factory of service adaptation manager can get guidance from the adaptation ontology on “what to do” and “how to do”; furthermore, the component description provides information on “what components can do the job” and “how to access them”. In addition, the self-regenerative adaptation service can flexibly swap to another implementation approach by replacing component description and those components bound with specific standards and technologies. At the same time, the high-level abstraction of self-regenerative adaptation and the software factory structure can still remain the same without being infected.

4 Implementation of Self-Regenerative Adaptation Knowledge

The three-layer adaptation knowledge has been implemented into separated meta-data documents (in format of XML). These machine-interpretable texts can flexibly integrate with the design of service adaptation manager on processes composition for providing self-regenerative adaptation service tailored to consumers' needs.

4.1 Adaptation Ontology Representation

As there is no “Correct” way or methodology for ontology development, this work is not trying to define a new ontology description language to address all the issues that an ontology development may need to grapple with. In order to quick test the research idea mentioned in this paper, it adopts a self-defined schema to describe adaptation process models instead of using standard technologies as DAML-S [12] or OWL-S.

Abstract process concepts defined in the adaptation ontology are mapped to the adaptation functionalities provided in the polyarchical middleware. This ontology describes the self-regenerative adaptation process as three developed integrated processes such as service discovery, service match making, and adapter construction. More specifically, a service lookup process consists of two separated sub-processes, namely both local and remote registry discovery. Also, an appropriate service match may consider several aspects such as the service name, and its programmatic signature (service signature). For the service parameters, it is also important to ascertain the matching of the number of parameters, their sequence and their data types are also important to ascertain. Based on this information, service consumers would then know the general characteristics for the self-regenerative adaptation service provided by the polyarchical middleware. They can then compose their own adaptation process models if they don't want to stick with the default one (offered by the polyarchical middleware). Indicated by the customized process models, the software factory of service adaptation manager can produce an adaptation service tailored to the consumers' special requirements. The details of the specific middleware components bound with those abstract processes will be covered in the middleware component description file.

4.2 Middleware Component Description

As we known, WSDL (Web service description language) has been widely adopted to specify the technical details of accessing service components, which have mainly to do with protocol type, message formats, serialization, and transport etc. Currently, the syntax of WSDL has only been developed to bind with Web service relevant technologies. The middleware component description here has to adopt a self-defined XML schema to specify information raised by various other service invocation protocols besides Web services. With the similar goal as WSDL does, this middleware component description provides the details of accessing those middleware components bound with specific implementation. It also provides the mapping from (abstract) processes defined in the adaptation ontology representation to those middleware components. In future, this work can be easily swapped with any particular specification language if it is designed to describe information about various service bindings with strong industry backing.

5 Conclusions And Future Work

This paper first briefs a description of an alternate novel adaptation approach via runtime self-regenerative adapters provided by the polyarchical middleware. The rest of the paper highlights the motivations and requirements for the application of a knowledge-centric model on supporting and reasoning the runtime generation of self-regenerative adaptation. Future evaluation of this work is underway including a number of further developments including: the design of an evaluation tested for this ontology model based on the developed polyarchical middleware; supporting the

semantic interpretation on the adaptation field across domains of same interests; another ontology model to describe the generation of adaptation template class, which might be transformed to deployable code (template class) encapsulated with specific information of service bindings and technologies in case the polyarchical middleware temporarily hasn't got one.

References

1. J. Kephart, D. Chess.: The Vision of Autonomic Computing, in Computer Magazine, IEEE, (2003).
2. Badger, Lee.: Self-Regenerative System (SRS) Program Abstract. DARPA, (2004).
3. Liang-Jie Zhang, Jen-Yao Chung, Researcher, IBM T.J. Watson Research Centre.: Developing Grid Computing Application: Part1, Introduction of a Grid architecture and toolkit for building Grid solutions, (2003).
4. Wassenberg, Wade, Software Engineer, and Motorola ISD.: Protocol Independent Programming Using Openwings Connector Services, (2003).
5. Heuvel, Willem-Jan van den. Matching and Adaptation: Core Techniques for MDA-(ADM)-driven Integration of new Business Applications with Wrapped Legacy Systems.
6. Furmento, Nathalie, Jeffrey Hau, William Lee, Steven Newhouse, and John Darlington.: Implementations of a Service-Oriented Architecture on top of Jini, JXTA and OGSA, In Proceedings of UK e-Science All Hands Meeting, (2003).
7. M.Yu, A. Taleb-Bendiab, D. Reilly.: A Polyarchical Middleware for Self-Regenerative Invocation of Multi-Standard Ubiquitous Services, In Proceedings of the IEEE International Conference on Web Services, (2004).
8. M.Yu, A. Taleb-Bendiab.: Generic Programming for Self-Regenerative Invocation cross-Standards, In Proceedings of 5th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking & Broadcasting, (2004).
9. David Llewellyn-Jones, Madjid Merabti, Qi Shi, Bob Askwith.: A Security Framework for Executables in a Ubiquitous Computing Environment, In Proceedings of Globecom, (2004).
10. Noy, Natalya F. and Deborah L. McGuinne.: Ontology Development 101: A Guide to Creating Your First Ontology, (2001).
11. Andreas Maier, Hans-Peter Schnurr, York Sure.: Ontology-based Information Integration in the Automotive Industry, In Proceedings of the 2nd International Semantic Web Conference, (2003).
12. Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David Martin, DrewMcDermott, Sheila A. McIlraith, and Massimo Paolucci Srin Narayanan, Terry Payne, and Katia Sycara.: DAML-S: Web Service Description for the Semantic Web.