Aspect oriented reflective middleware model for dynamic adaptation

S.Suganthi[1], R.Nadarajan[2]
[1]Department of Computer Technology and Applications
Coimbatore Institute of Technology
Coimbatore 641 014, India.

[2]PSG College of Technology
Coimbatore 641 014, India.
{suganthi_cit, nadarajan_psg}@yahoo.co.in

ABSTRACT: *In a distributed environment, making a system dynamically adaptable to the fine-grained changes in the functionalities of the system is a big challenge in the software industries. Currently, the solution for adaptation is achieved using reflective middleware. Since reflective middleware is not an efficient solution for dynamic adaptation, in this paper a solution is proposed by combining aspect oriented approach with reflective middleware. In this proposed aspect oriented reflective middleware model, fine grained changes in the functionalities are captured as 'dynamic aspects' that are dynamically weaved with application components on the basis of point-cut specification. This model is implemented using Reflection pattern, Aspect oriented approach and Remote Method Invocation communication model. This model is analyzed using the factors such as programmatic complexity and execution time. The novel approach used in this model increases the possibilities of incorporating the changes without affecting the core functionalities of the system and reduces the performance overhead.*

Keywords: Aspect-Oriented Reflective Middleware, Aspect- Oriented Approach, Dynamic Adaptation, Reflective Middleware, Remote Method Invocation

Received: 12 May 2010, Revised 19 June 2010, Accepted 28 June 2010

1. Introduction

To be in harmony with continuous variations of the execution environment and user needs in distributed system, dynamic adaptability is to be introduced in the system at the architecture level. Adaptive middleware is considered as an optimal solution to introduce dynamic adaptability at the architectural level (cited in Sam Malek et al., 2006). Many research works have focused on creating adaptive middleware using reflective pattern.

Reflective Middleware enables changes in the system behavior by adapting core services specified in the software without any external modification. Reflection means that all reasoning at the meta-level is performed by the component itself. This is achieved through self discovery objects, which perform various types of reflection such as structural reflection, behavioral reflection, resource reflection and discovery (cited in Duncan Pemberton).

It has been observed that, several researches on reflective middleware systems have been focused on defining the reflective components for performing both the introspection and adaptation and are implemented on the specific component models (cited in Nikos Parlavantzas et al., 2000). This makes the component as heavy weight component, which reduces the efficiency of the middleware. It can be addressed through modularizing the reflective middleware such as aspectizing the reflective components (cited in Nelio Cacho et al., 2006).

This paper presents the aspect oriented reflective middleware architecture as a solution to improve the adaptability of existing components to runtime needs. In this model, the performance overhead and complexity of the reflective middleware are reduced by separating the process of specifying the changes in the functions of the component using separation of concerns and making the component to change its behavior using aspect weaving and structural reflection mechanism. In this proposed solution, the dynamic requirements are considered as cross-cutting concerns, defined as advices and weaved based on point-cut expressions. This strategy is used to enrich the existing functionalities of the system to dynamically adapt the changes and perform the action accordingly.

The key benefit of this approach is to increase the dynamic adaptation efficiency of the system without affecting the existing component structure and behavior. The key contributions are specifying the fine grained functional changes as aspects and performing the dynamic weaving using reflection with distributed components. The proposed model is evaluated by analyzing the programmatic complexity and execution overhead.

2. Aspect Oriented Reflective Middleware Architecture

This section describes the middleware architecture designed and organized using reflective infrastructure and aspect oriented approach for incorporating dynamic adaptability feature over the existing architectural components.

2.1 Approach and Design Principles

The basic approach used in this model is used to structure the middleware with aspect and reflective layers and place it on top of the application component layer. The following are the design principles used to design the model.

- Remote Method Invocation (RMI): In this model, the dynamic class loading feature of RMI is used. The dynamic class loading enables the dynamic change in the behavior of the function.
- Reflection: Reflection is defined as self discovery, which is used to introspecting the interfaces, methods, their arguments and return types at the meta level by the component itself. This principle is used to design the meta layer of this model.
- Aspect Oriented Approach: The feature of defining cross-cutting concerns into aspects to retain modularity allows for the clean isolation and reuse of code addressing the cross-cutting concern. The above mentioned feature is used to handle the fine grained changes in the services of the component.

Component oriented architecture is used to make the middleware as a reusable model.

2.2 Architectural Overview

The Figure 1 depicts the architectural design of the middleware, which consists of aspect-layer and meta-layer. The functions of aspect layer are to extract the request received in the xml format, generate them as cross cutting concerns and weave the cross cutting concerns at various joint points in the components. Here all requests for dynamic changes in the requirements are considered as cross cutting concerns. Meta layer does the component introspection and provides details about the interfaces and methods of the component.

2.3 Architectural Elements

The class structure of the middleware is shown in the Figure 2 and the functionalities of the classes and their associations are elaborated below.

AspectManager: The AspectManager receives the request from the clients and initiates the activities such as xml parsing, aspect creation, dynamic method invocation and dynamic weaving.

XMLExtractor: The XMLExtractor receives the xml file that consists of dynamic request for incorporating the minor variations in the functionalities of the system and extracts the details such as component name, interface name, function name and details about the fine-grained changes requested by the client using XML parser.

AspectGenerator: The AspectGenerator receives the name of the interface and function to be refined and also details about the run-time functional changes requested by the client from XMLExtractor. It describes the functional changes specified by the user as aspects and creates
the aspect with point-cuts and advices to implement the logic associated with the dynamic change specification and the joint-point specification.

ComponentIntrospector: The ComponentIntrospector receives the component name from the XMLExtractor. It makes use of the reflection logic to extract the structural composition of the component such as interface classes and method signatures.

PointcutMethodWrapper: It dynamically loads and executes the method specified in the point-cut expression. It does the

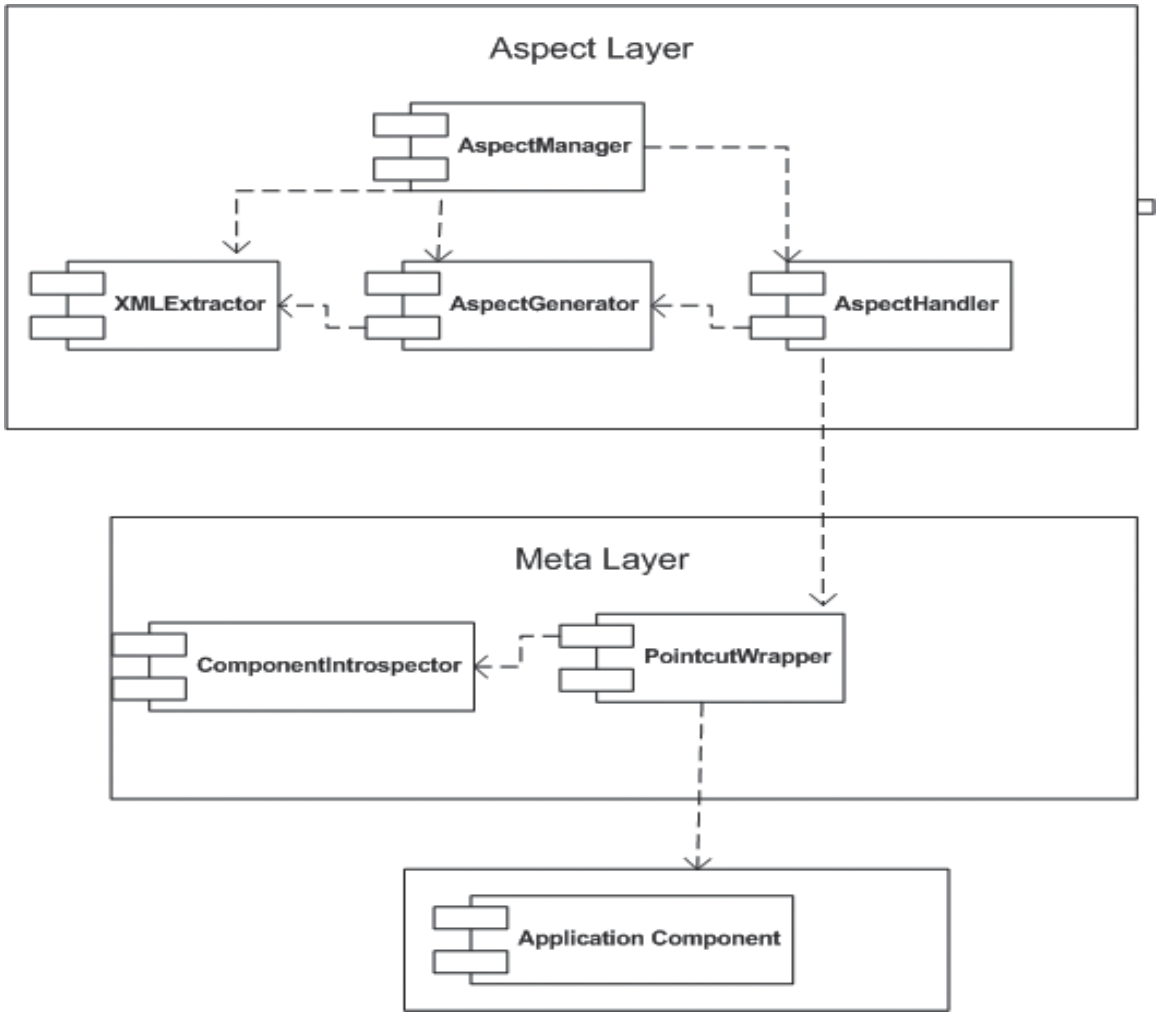

Figure 1. Aspect Oriented Reflective Middleware

dynamic execution through performing the operations such as receiving the method signatures for the methods specified in the point-cut expression from ComponentIntrospector and executing the invoke() method call.

AspectHandler: The AspectHandler gets the Aspect class from AspectGenerator and performs load time weaving with wrapper method defined in the PointcutMethodWrapper.

2.4 Behavioral Model

```
<?xml version=”1.0"?>
<CHANGEREQUEST>
 <COMPONENT>
      <INTERFACENAME>IArithmeticService</INTERFACENAME>
 </COMPONENT>
 <OPERATION>
      <OPERATIONNAME>arithmeticOperation</OPERATIONNAME>
 </OPERATION>
 <CHANGESPECIFICATION>
      <SPECIFICATION>Validate the inputs</SPECIFICATION>
      <JOINTPOINT>Before</JOINTPOINT>
 </CHANGESPECIFICATION>
</CHANGEREQUEST>
```

Figure 2.1 Validation Request Specification XML

Here illustrated the scenario of performing the validation in the arithmeticOperation function of the ArithmeticService component. Validating all the inputs before performing the arithmetic operation is the change to be incorporated into the arithmeticOperation function specified in the IArithmeticService interface. The Figure 3 illustrates the sequences of operations involved in making the application to adapt the validation request specified in the Validation Request Specification XML format as stated in Figure 2.1.

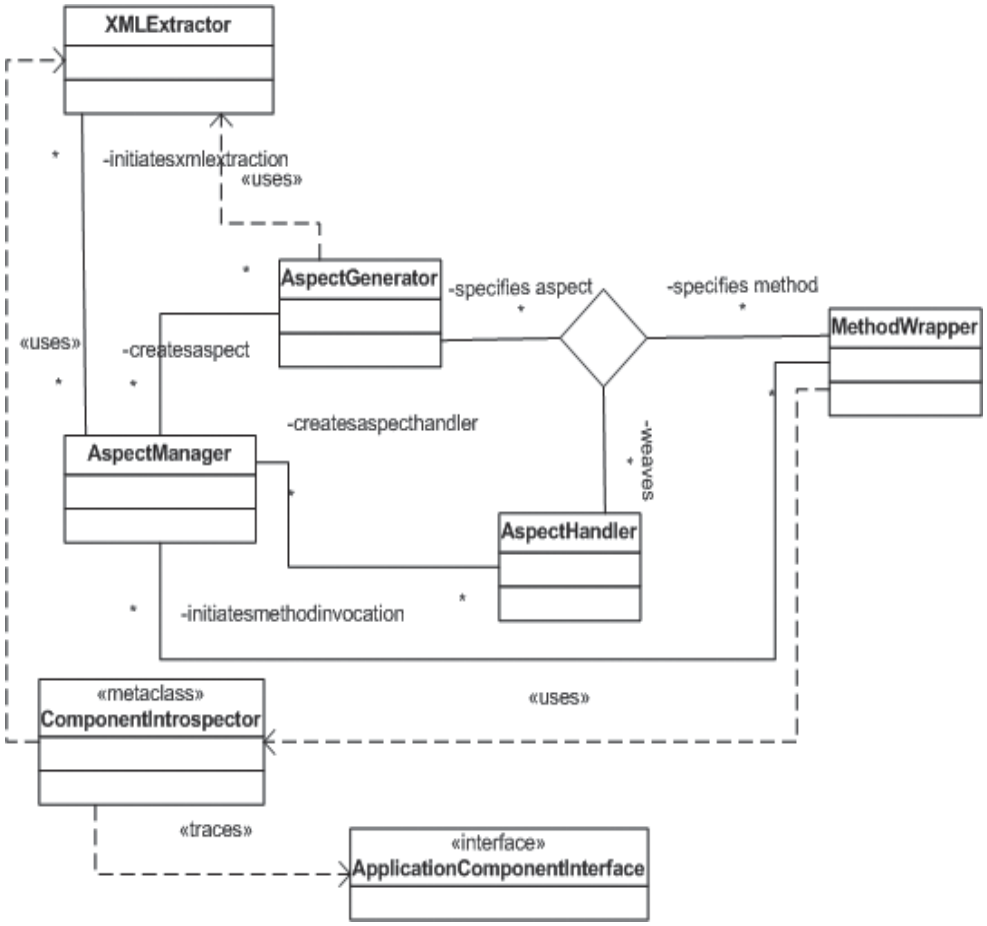

Figure 2. Static Structural View of the Middleware

3. Performance Evaluation

The aim of this work is to increase the adaptability. This cane be achieved by increasing the Dynamic Request Hit Ratio (DRHR). To increase the request hit ratio, the overheads of the proposal such as Re-Configuration and Execution Time (RCET) are analyzed and solved. To make an efficient adaptable system, it is required to increase DRHR and reduce the RCET.

Hence, Dynamic Adaptability = DRHR / RCET
Where RCET = Time taken for component discovery x Time taken for weaving and executing the aspect

In this middleware, use of separation of concerns mechanism to separate the logic for incorporating changes in the functionality of the component; allows all fine grained changes are easily incorporated over the operations defined in the component.

Time taken for component discovery is based on the component framework and is negligible. The aspect weaving and execution time is calculated using the programmatic complexity of the cross cutting concern and execution time of the function along with the aspect.

The programmatic complexity is measured with the number of lines of code required to specify the composition of aspect code with the underlying component, which is always O(1).

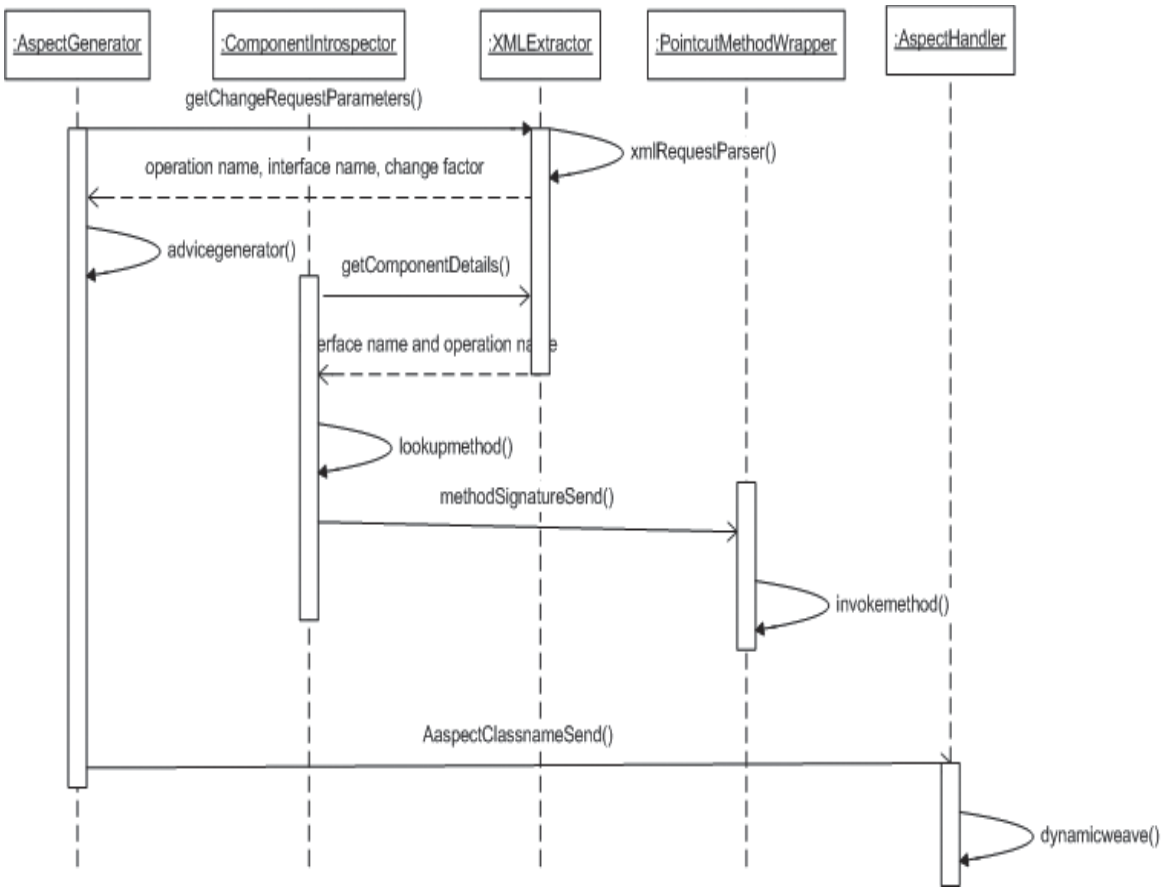

Figure 3. Functional Changes Invocation Scenario

The execution time is analyzed as follows:

- Time taken for executing an operation (method) in a component is calculated.
- Time taken for executing a method after incorporating the changes (with added lines of code) and without using aspect approach is calculated.
- Defining the changes in the method as an aspect and time taken for executing the method along with static and dynamic advices is calculated.

Sampling Method Type	Execution time in milliseconds			
	Incorporating Changes in the Method			
	Base Method	Without using Aspect	Using Static Aspect	Using Dynamic Aspect
Method without arguments	392	395	374	372
Method with arguments	404	393	379	377

Table 1. Performance Data

From the observed result as shown in the Table 1, it is identified that the execution overhead is reduced by 5.6% using static advice and 6.1% using dynamic advice for incorporating changes in the operation (method) without any argument of the component.

The same analysis is done for a method with arguments and observed that the execution overhead is reduced by 3.6% using static advice and 4.2% using dynamic advice. The analysis results are clearly specified in the Figure 4.

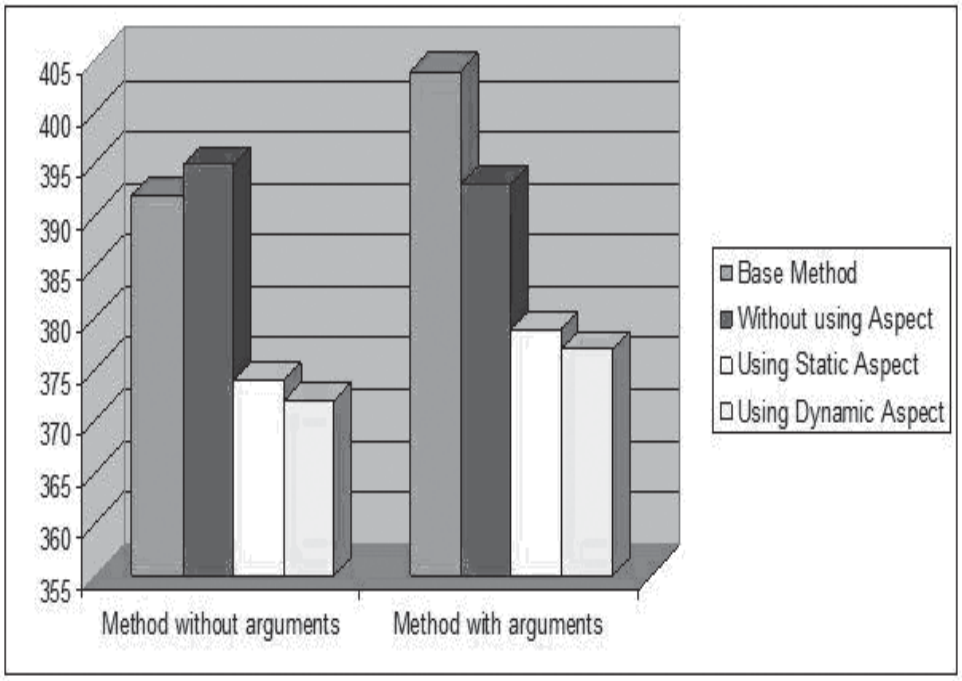

Figure 4. Performance Analysis Chart

Finally it is concluded that incorporating the fine grained changes in the system as aspects reduces the execution overhead.

4. Related Work

There are many researches done on adaptability using reflective middleware, aspect oriented approach and component frameworks in different ways. In the work(cited in Bholanathsingh Surajbali et al., 2007), the aspect oriented support layer is placed on top of the OpenCOM reflective model and distributed aspect oriented services are layered over the GridKit middleware, which allows dynamic composition of dynamic aspects. Unlike our model, the overhead incurred while executing a function with advice over the baseline function without advice is slightly high. Also the performance of this model is confined with the limitations of OpenCOM and GridKit frameworks. The other works (cited in Nelio Cacho et al., 2006, Sam Malek et al., 2006 and Duncan Pemberton) concentrated on making the existing middlewares like COM and ORB as reflective middlewares and aspectizing the reflective components and not focused on aspectizing the changes in the functionalities of the components.

In our approach, adaptive middleware is a distributed component framework build with aspect oriented approach, reflection model and remote method invocation communication model which is portable and compatible with any application components. Our main contribution is to capture the changes in the functions of the component as aspects, generate the advices dynamically using adaptation rules and dynamically weave the aspects into the target functions that are identified using meta-information provided by the meta-layer. The users can submit the changes in the functionalities using XML format to make it (language independent) accessible with any application. Maintaining the separate layers for capturing requirement changes as aspects and defining pointcut expression and advices in aspect layer and introspecting the information about application components in meta layer, which imposes the distribution. Remote Method Invocation is used for component discovery and communication.

5. Conclusion and Future Work

In this paper, middleware model with aspect and meta layers is used to efficiently adapt the fine grained changes on the functions of application components with less performance overhead. The benefits of defining changes in the functionalities of the components as aspects and dynamically loading the functions using reflection and remote method invocation in the middleware enables the system to dynamically adapt the functional changes.

As future work, it is proposed to integrate this middleware architecture with OSGi framework to develop web service components with dynamic adaptation feature. Another enhancement is to create the middleware model using Model Driven Architecture, which enables this middleware can be easily mapped with any application component model.

References

[1] Surajbali, Bholanathsingh., Coulson, Geoff., Greenwood, Phil., Grace, Paul (2007). Augmenting Reflective Middleware with an Aspect Orientation Support Layer. *In*: Proceedings of the 6th Workshop on Adaptive and Reflective Middleware, ARM 2007, ACM/IFIP/USENIX International Middleware Conference, Newport Beach, CA, USA, 2007. ACM Publications.

[2] Cacho, Nelio., Batista, Thais., Garcia, Alessandro., Anna, Claudio Sant., Gordon Blair. (2006). Improving Modularity of Reflective Middleware with Aspect-Oriented Programming, *In*: Proceedings of the 7th Workshop on Software Engineering and Middleware, Portland, 2006. ACM Publications.

[3] Malek, Sam., Seo, Chiyoung., Ravula, Sharmila., Petrus, Brad., Medvidovic, Nenad (2006). Providing middleware –Level Facilities to support Architecture – Based Development of Software Systems in Pervasive Environments. *In*: Proceedings of the 4th International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC 2006), Melbourne, Australia, 2006. ACM Publications.

[4] Bencomo, Nelly., Sawyer, Pete., Grace, Paul., Blair., Gordon (2006). Ubiquitous Computing: Adaptability Requirements Supported by Middleware Platforms. *In*: Proceedings of the Workshop on Software Engineering Challenges for Ubiquitous Computing, Lancaster, UK.

[5] Couto, Ricardo., Rocha, Antunes da., Antunes da Rocha, Endler, Markus (2006). Middleware: Context Management in Heterogeneous, Evolving Ubiquitous Environments, IEEE Distributed Systems Online 7 (4) 1541- 4922, IEEE Computer Society.

[6] Frei, Andreas Ralph, (2005). Jadabs–An Adaptive Pervasive Middleware Architecture. Doctor of Technical Sciences thesis, submitted to Swiss Federal Institute of Technology, Zurich.

[7] Pemberton, Duncan. RComponents - Reflective & Adaptable Components. http://www.comp.lancs.ac.uk/computing/research/cse.

[8] Hveding, John Christian (2005). An Aspect-Oriented Approach to Adaptive Systems. Master thesis, 2005.

[9] Nelly Bencomo, Gordon Blair, Paul Grace The world is going MAD: Models for Adaptation. www.comp.lancs.ac.uk ~bencomo/publications.html.

[10] Parlavantzas, Nikos., Coulson, Geoff., Clarke Mike., Blair. Gordon (2000). Towards a Reflective Component-based Middleware Architecture. *In:* Proceedings of the Workshop on Reflection and Metalevel Architectures, Sophia Antipolis and Cannes, France.

[11] Bass, Len., Clements, Paul., Kazman, Rick (2003). Software Architecture in Practice. Second Edition, Pearson Education.

[12] Szyperski, Clemens. (1999). Component Software: Beyond Object Oriented Programming. First Edition, Addison-Wesley.