Middleware Model for Adapting Dynamic Requirements

S.Suganthi[1] , R.Nadarajan[2]
[1] Department of Computer Technology and Applications
Coimbatore Institute of Technology
Coimbatore 641 014, Tamilnadu, India
suganthi_cit@yahoo.co.in
[2] Department of Mathematics and Computer Applications
PSG College of Technology
Coimbatore 641 015, Tamilnadu, India
nadarajan_psg@yahoo.co.in

JDIM
Journal of Digital Information Management

ABSTRACT: *Making a distributed system adaptable to the dynamic variations in the user requirements is a big challenge in software industries. It is identified that the solution for the adaptability of such dynamic variations can be achieved through reflective middleware. In this paper an efficient solution is proposed by combining aspect oriented approach with reflective middleware. The significant feature of the proposed aspect oriented reflective middleware model is capturing the dynamic variations in the functionalities of the system as aspects. The three major components defined in this model to implement dynamic adaptability are Aspect Generator, Aspect Weaver and Remote Method Invoker. Aspect Generator component is used to define dynamic requirements as aspects. Aspect Weaver provides service to weave the dynamic requirements with the application components associated with that requirement. Remote Method Invoker is the meta layer component, which provides the platform for the Aspect Weaver to execute its service. The metric to measure the efficiency of the system adaptability has also been proposed here. The adaptability approach implemented through this model has been evaluated and proved as an efficient solution using this metric. This model has also been evaluated at an architectural level using formal analysis to prove that the dynamic requirements can be incorporated without affecting the existing functionalities of the system.*

Categories and Subject Descriptors
D.2.1 [Software Requirements/Specifications]; Tools: **D.2.4 [Software/Program Verification**]; Formal methods

General Terms: Middleware models, Distributed Systems, Software user requirements

Keywords: Aspect-Oriented Reflective Middleware, Aspect-Oriented Approach, Dynamic Adaptation, Reflective Middleware, Dynamic Requirements, Adaptability Metric

Received: 10 June 2011, Revised 13 August 2011, Accepted 28 August 2011

1. Introduction

Adaptability is emerging as a crucial enabling capability for many applications, which are deployed in dynamically changing environment [9]. Nowadays, most of the software maintenance activities are projected on making the system adaptable to the dynamic variations in the user or environment needs. The effort expended to perform such adaptive maintenance can be reduced by introducing dynamic adaptability feature in the system. The dynamic adaptability is defined as the ability of a software system to adapt the behavioral or structural changes that occur either internally or externally in the operating / user environment while the system is running [12]. Several approaches exist to achieve dynamic adaptability are intertwining adaptation with application behavior, application independent middleware and adaptation based middleware [14] [13]. Reflective adaptive middleware designed using a reflection mechanisms is identified as a better solution for making the system to adapt themselves to run-time needs [4] [5].

A reflective architecture logically models the adaptable system with two layers namely base-layer and metalayer. The meta-layer of the reflective system is responsible for supervising the occurrence of the run-time needs to be adapted and making the system to adapt the run-time needs. It implies that the components in the meta-layer of the reflective system are performing both the reflection and adaptation [9], which reduces the adaptability of the system. This major issue of the reflective system can be addressed through modularizing the reflective middleware that is specified as aspectizing the reflective components [2].

Another issue in the reflective middleware implemented using object oriented paradigm is not providing the proper means to achieve the required level of localizing the

reflection-specific concerns. Introducing aspect oriented technology in the development of reflective based adaptive system can address the issue of localization [8]. Aspect oriented paradigm is a new approach to software development that addresses limitations inherent in other approaches, including object-oriented programming. Aspect oriented approach modularizes the concerns spread over the system, which is named as cross cutting concerns [15]. Crosscutting concerns are encapsulated in separate modules, known as aspects, so that localization can be promoted. It shows that aspect oriented software development provides a means to modularize the crosscutting concerns in software systems [20], which results in reducing the development, maintenance and evolution costs. Many of the existing reflective middleware models are created using traditional middleware models, which are suffering with the limitations of managing the meta-level information of the components [22].

This paper presents the aspect oriented reflective middleware model as a solution to improve the adaptability of the existing components to runtime needs. In general, aspects are used for representing logging, tracing, debugging, security and program verification issues [18]. In this work, aspects are used for representing the run-time changes in the user requirements, which are specified as dynamic requirements. In this model, the complexity of the reflective middleware is reduced by separating the process of adapting run-time needs from the reflection. The features of the proposed model are (i) considering the dynamic requirements as aspects, (ii) defining meta-layer to perform the introspection and (iii) defining the aspect-layer to create aspects and to weave aspects with the methods associated with the run-time requirements through meta-objects defined in the meta-layer of the system. These features of the proposed model increase the dynamic adaptability efficiency of the system without affecting the existing component structure and behavior. The formal analysis and experimental analysis are carried out to prove the adaptability efficiency of the proposed model.

2. Background

The proposed middleware model is designed using reflective architectural pattern and aspect oriented paradigm and are described as follows.

2.1 Reflection

Reflection is the ability of a system to watch its computation and possibly change the way it performs.

A reflective architecture constituted with base-level and meta-level layers, which is allowing the system to change its behavior and structure in accordance with external events by itself. The base-level is the system that should be made self-adaptable and meta-level reifies the base level information and plan its evolution when particular events occur [5].The meta-level is connected with base-level through the data structure, which maintains all the characteristics of the base-level.

Reflection is a self discovery, which represents that all reasoning at the meta-level is performed by the component itself and it is categorized as structural reflection, behavioral reflection, resource reflection and discovery reflection [7]. Reflective middleware performs the changes in the system through meta-objects that are represented as self-discovery objects in the meta-level. These self discovery objects are responsible to perform the introspection of 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.

2.2 Aspect Oriented Approach

The feature of aspect oriented approach is to encapsulate the cross-cutting concerns into aspects, which redefines modularity and allows for the clean isolation and reuse of code addressing the cross-cutting concern. The concern is a function, which is to be considered as a secondary operation in many functions of the system and named as a cross-cutting concern. Cross-cutting concern is encapsulated in a separate module known as aspect. The code associated with the cross-cutting concerns along with the joint-point in the functions is known as an advice. The specification of the cross-cutting concern and the functions into which the cross-cutting concern is to be added (joint-points) is expressed as the point-cut. Finally an aspect is designed as a composition of point-cut and advice. This modularization process reduces the software development, maintenance and evolution costs. This feature is used to separate the handling of dynamic requirements from the application. The benefits of aspect oriented programming can be ushered onto middleware frameworks to improve the modularity and to reduce the development cost of the enterprise applications [23].

3. Related Work

There are many works done on dynamic adaptability using component based reflective middleware and aspect oriented approach.

The research work stated in the paper[1] focused on augmenting the reflective middleware with an aspect orientation support layer for better modularization of services provided by OpenCOM, a component model and GridKit, an OpenCOM based middleware . In this model, the components of the aspect oriented support layer placed on top of the OpenCOM reflective layer and the distributed aspect oriented services like aspect composition are placed over the GridKit middleware. This design approach shows that the adaptability provided by this model should be confined with the limitations of OpenCOM and GridKit frameworks. The AspectOpenCOM, a reflective component model [19] provides a meta-object protocol capable of fine-grained adaptation of deployed aspects and suffers with the limitations of meta-object protocol.

The work presented in the paper [2] provides the solution to aspectize certain category of cross-cutting concerns such as logging, monitoring and controlling the changes in the states of the resources on OpenORB reflective model.

This model improves the adaptability of the OpenORB based systems. The work stated in [7] concentrated on introducing reflective components at ORB level to create re-configurable execution container and modularizing the reflective components using aspects.

The research works discussed above were concentrated on (i) introducing the adaptability feature on traditional middleware models like COM, CORBA for adapting the run-time changes of the services provided by them; (ii) aspectizing the reflective components defined in OpenCOM and OpenORB reflective middleware models and (iii) specifying cross-cutting concerns as aspects.

These works did not focus on (i) creating a model, which can be integrated with any component models; (ii) specifying run-time requirements as aspects and (iii) providing the solution to adapt the dynamic requirements without affecting the dependent components.

In our approach, adaptive middleware is created as a distributed component framework, which is portable and compatible with any application component model using aspect oriented approach and reflection architectural pattern. Our main contribution is to capture the dynamic requirements associated with the functions of the component as aspects and dynamically weave the aspects into the target functions using meta-information provided by the metalayer. The users can submit the changes in the functionalities using XML format (language independent) to make it accessible with any application. In this model the layer defined for capturing requirement changes as aspects, defining point-cut expression and advices is separated from the layer defined for performing the application components introspection, which makes the middleware as a distributed one. Remote Method Invocation is used for component discovery and communication.

4 Middleware Model Structure

4.1 Design Principles

The proposed middleware model is based on the suggestion in [3], which states that providing an architectural middleware described with the key architectural abstractions to support dynamic adaptation

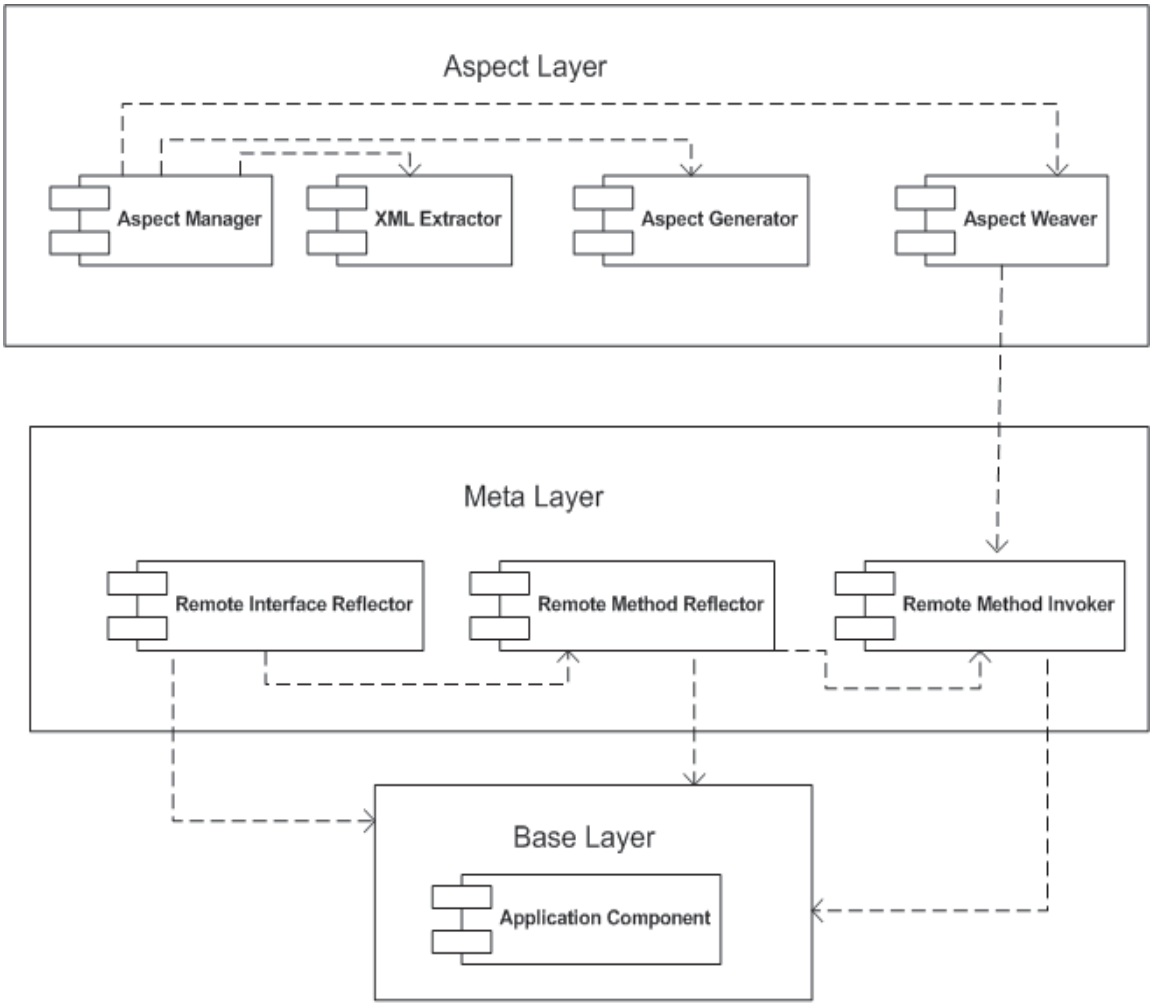

Figure 1. Aspect Oriented Reflective Middleware Architecture

facility has been suggested as a better solution rather than providing the solution using traditional middleware platforms. In this middleware model the meta-layer is redefined with the following functions: (i) to provide the interface for performing the component introspection and (ii) to provide the wrapper for invoking only the methods in the component, which are associated with the run-time changes and separating the adaptation function from the meta-layer and specifying it in the aspect-layer .The function of aspect-layer is to implement the run-time requirements as aspects and weaving them with wrapper methods. The design principles to design the aspect as stated in [24] are followed to design the aspects in this paper. By combining aspects and reflection, this middleware model is considered as a promising approach to improve usability, extensibility, modularity and customization capabilities of middleware platforms [21].

4.2 Architecture of the Proposed Middleware

The Figure1 depicts the architectural design of the middleware, which consists of Aspect Layer and Meta Layer. The Aspect Layer captures the request for variations in functionalities and generates them as Aspect objects. Meta Layer does the component introspection and provides the details about interfaces and methods of the component.

The following are the components in the Aspect Layer

• Aspect Manager – It coordinates all the activities such as parsing the XML request, generating the aspects to represent dynamic requirements and weaving the aspect with meta application component in the Aspect layer. The request for dynamic requirements is sent to Aspect Manager in XML format.

• XML Extractor – It parses the dynamic requirements request sent in the XML format and extracts the details such as components and functions associated with dynamic needs and also the requirements specification.

• Aspect Generator – According to the dynamic requirements specification, the advice named dynRequestMethod, which specifies the implementation of the run-time changes and point-cut, which specifies the method into which the advice is to add are defined in this class. In this model Wrapper method defined in the Remote Method Invoker is specified as a point-cut for all aspects. This makes the declaration of point-cut and advice in a general way.

• Aspect Weaver – It weaves the aspect defined in the Aspect Generator with Wrapper method of the function associated with the dynamic requirement.

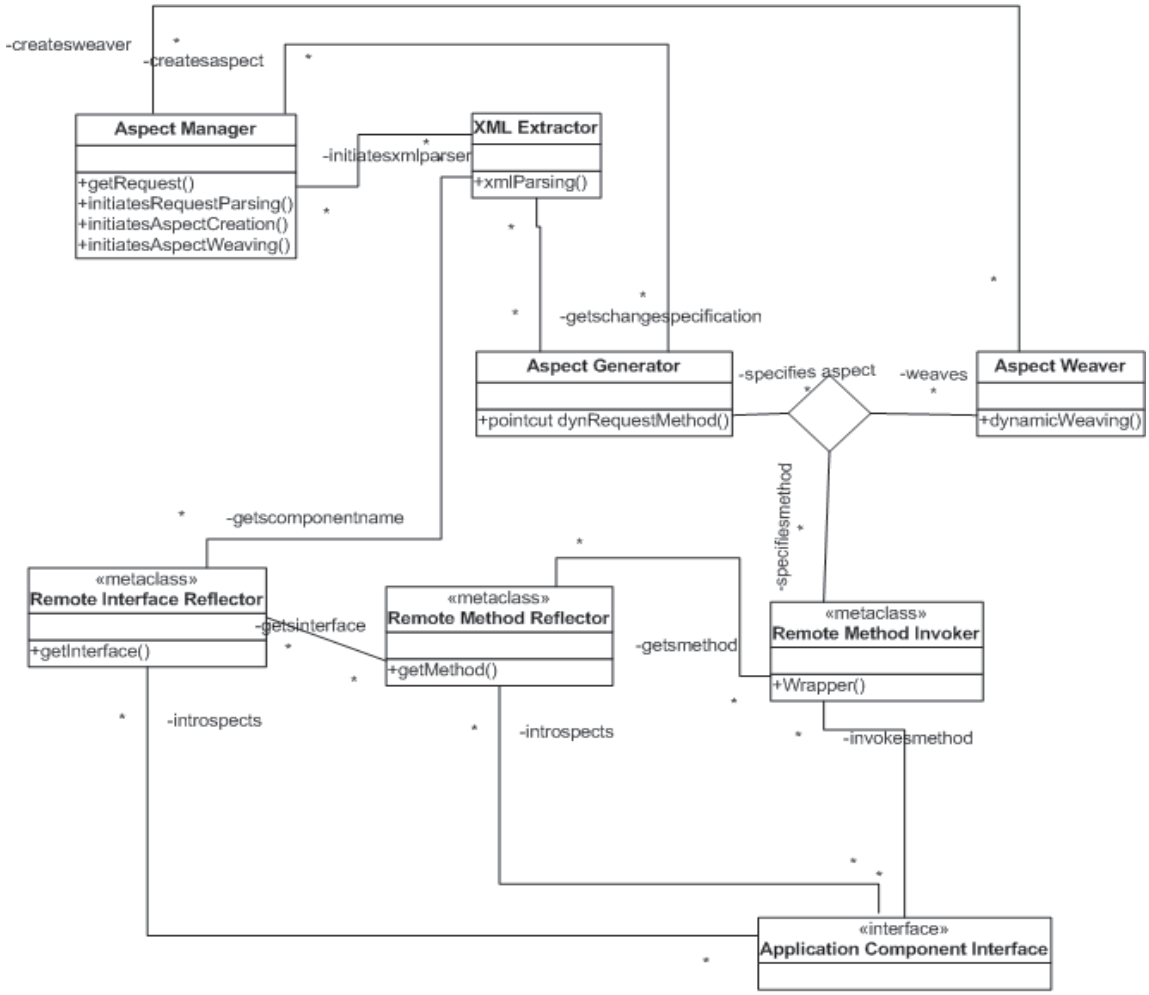

Figure 2. Class View of the Middleware

The components in the Meta Layer are,

• Remote Interface Reflector – It provides the service for extracting the structural details of the interface such as operations specified in the interface and the classes implementing those operations using introspection.

• Remote Method Reflector – It provides the facility forextracting the information about the methods specified in a particular class using introspection.

• Remote Method Invoker – It acts as wrapper to dynamically invoke the operation according to the dynamic needs.

The Figure 2 depicts the structure of the classes implementing the services offered by the components in the metalayer and aspect-layer.

5 Experiment

5.1 Model Implementation

This section describes the scenario of making the changes in the customer information verification procedure while opening an account in a bank. Opening an account in a bank is specified as one of the functions in the Banking Transaction service component. In some cases, the customer may produce the document which is not in the document list specified for the verification of the customer information while opening an account. Performing the customer information verification using the new document is defined as an aspect.

The dynamic request for performing the customer information verification using new document is given in the XML file as shown in the Listing 1. The procedure used to perform the customer verification using new document is specified in the dynRequestMethod advice of the Aspect Generator class as shown in the Listing 2. The specification of weaving the advice with wrapper method defined in the Remote Method Invoker is described in the dynRequestMethod point-cut of the Aspect Generator. The wrapper method acts as a wrapper for invoking the Opening Account method in the Banking Transaction component. The wrapper method is created using java reflection as shown in the Listing 3. The load time weaving mechanism is implemented in the dynamicWeaver method of the Aspect Weaver as shown in the Listing 4.

The Figure 3 illustrates the sequence of operations involved in making the component to incorporate the variation in the verification procedure. Remote Method Invocation (RMI) is used for the dynamic class loading which enables the dynamic change in the behavior of the function.

In this scenario, another aspect is defined for incorporating a new policy such as allowing the customers to perform the transaction for the amount exceeds certain limit only if they produce an identity proof issued by the government and repeat the process as specified in Figure 3. This new policy should be specified in all the functions associated with the transaction. Representing this new policy with

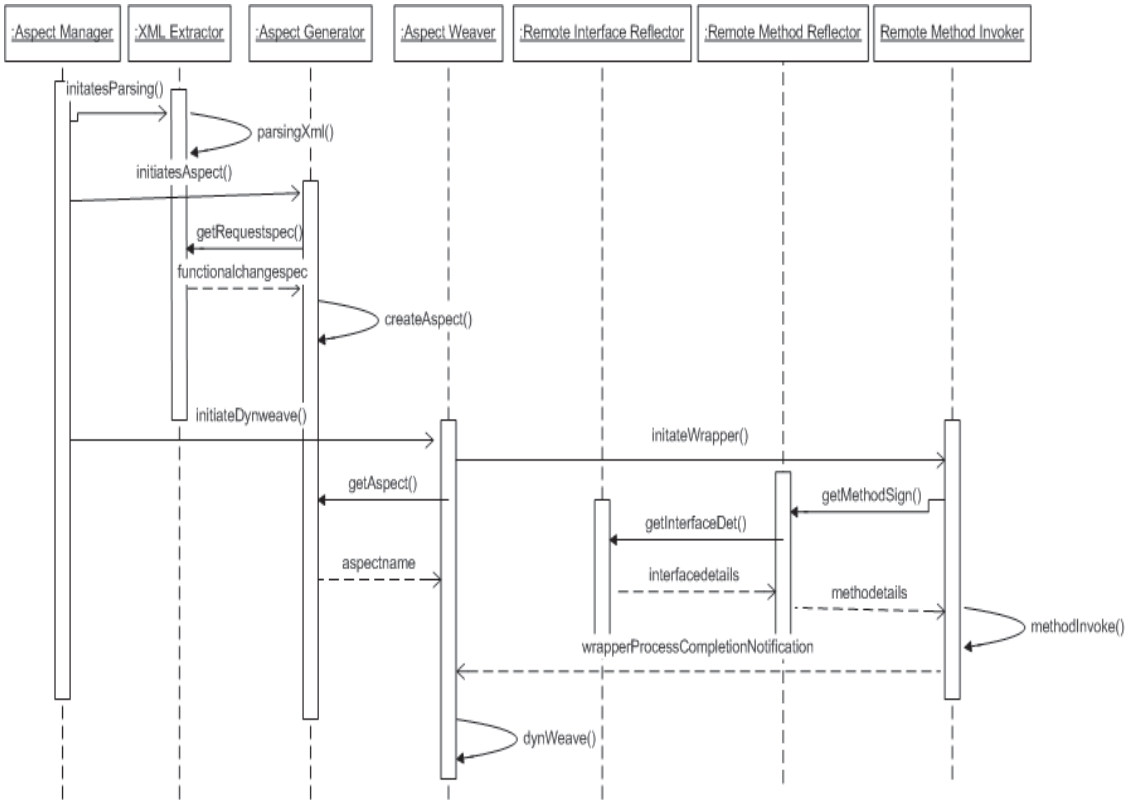

Figure 3. Functional Changes in the Customer Verification Procedure Scenario

aspect will make all the functions associated with the transaction to inherent this change by weaving this aspect with them.

```
<?xml version=”1.0"?>
<CHANGEREQUEST>
<COMPONENT>
   <INTERFACENAME>Banking Transaction Service</
INTERFACENAME>
</COMPONENT>
<OPERATION>
  <OPERATIONNAME>Opening an Account</
OPERATIONNAME>
</OPERATION>
<CHANGESPECIFICATION>
    <SPECIFICATION>Perform the verification using the
new document</SPECIFICATION>
   <JOINTPOINT>Before</JOINTPOINT>
</CHANGESPECIFICATION>
</CHANGEREQUEST>
```

Listing 1. XML Request for Changes in the Customer Verification Procedure

```
public aspect Aspect Generator{
        pointcut dynRequestMethod(): execution(public
void wrapper());
      before(): dynRequestMethod() {
                /* procedure to perform the customer
verification using new document */
                }
}
```

Listing 2. Aspect Generator Structure

```
public class Remote Method Invoker {
      public void Wrapper();
     /* procedure to dynamically invoke the verification
method on which the change specified in the aspect is to
be incorporated. */
}
```

Listing 3. Remote Method Invoker Structure

```
public class Aspect Weaver {
    public dynamicWeaver {
   /* command line execution of ajc Aspect Generator.aj –
outxml –outjar timing.jar;
   java –javaagent : c:/aspectj1.6/lib/aspectweaver.jar –
classpath “. ;
      timing.jar;c:/aspectj1.6/lib/aspectjrt.jar” Remote
Method Invoker; */
  }
}
```

Listing 4. Aspect Weaver Structure

5.2 Adaptability Efficiency Evaluation

In this work, a metric for representing the efficiency of the adaptability of the system has been introduced. The dynamic measurement specified in the AOP-HiddenMetrics framework [16] and empirical assessment of the impact of aspect oriented programming on software stated in [17] have been analyzed and derived the metric to represent the efficiency of the system adaptability. The adaptability efficiency of the system (AES) is measured using the number of methods of the components associated with the aspects. The complexity of the aspect is described using the number of point-cuts and advices in the aspect.

The number of methods associated with an aspect is specified as the conceptual binding between the aspect and the methods.

Let us assume that the aspects are represented as set $A = \{a_1, a_2,\ldots,a_n\}$, the set of components are represented as $C = \{C_1, C_2, \ldots,C_n\}$, the methods in a component is specified as $M_i = \{ m_{i1}, m_{i2},\ldots,m_{in}\}$, where $M_i \varepsilon\ C_i$ and $C_i \varepsilon\ C$ and the set of methods in the various components of the system is specified as $M = \{ M_1, M_2, \ldots, M_n \}$.

Then the conceptual binding between the aspect and the methods in the components is specified as *CBAM(a, m)* where $a \varepsilon\ A$ and $m \varepsilon\ M_i$.

$$CBAM(a, M) = \frac{\sum_{i=0}^{n} (CBAM(a, M_i))}{\sum |M_i|}$$

where $a \varepsilon A$, $m_i \varepsilon\ M_i$ and $CBAM(a, M_i)$ represents the number of methods in a component C_i is associated with the aspect ‘*a*’

The adaptability efficiency of the system *(AES)* is measured using the conceptual binding between the aspects and the methods in the system.

$$AES = \frac{\sum_{i=0}^{n} (CBAM(a_i, M))}{|A|}, \text{ where } a_i \varepsilon\ A$$

It implies that the adaptability of the system is increased with the number of methods associated with the aspects.

The evaluation of the adaptability efficiency of the Banking Application designed using this middleware model is described as below:

The Banking Application stated here, is comprised of the Banking Transaction, Authentication and Customer Service Components. The Banking Transaction Component is defined with the methods to create different types of account and to perform deposit, withdrawal, bill payment, fund transfer, loan payment transactions on the account.

The set ‘C’ represents the components of the banking application.

C = {Banking Transaction Component, Authentication Component, Customer Service Component}.

Component C_1 = Banking Transaction Component

The set "'M_1" represents the methods in the component where $M_1 = \{m_{11}, m_{12}, m_{13}, .., m_{110}\}$.

Number of methods in a component C_1 that is $|M_1|$=10

The set $A = \{a_1, a_2\}$ represents the dynamic requirements associated with the Banking Transaction Component of the banking application, where a_1, a_2 are aspects.

Aspect a_1 = Changes in the customer verification procedure

Aspect a_2 = Changes in the deposit policy

Number of dynamic requirements or aspects $|A|$= 2

Number of methods in the component C_1 associated with the aspect a_1 = 7

$CBAM(a_1, M_1) = 7 / 10$, where $a_1 \varepsilon A$

Number of methods associated with the aspect a_2 = 5

$CBAM(a_2, M_1) = 5 / 10$, where $a_2 \varepsilon A$

$AES = CBAM(a_1, M_1) + CBAM(a_2, M_1) / 2 = 0.6$

The following chart illustrates adaptability efficiency evaluated using the above data.

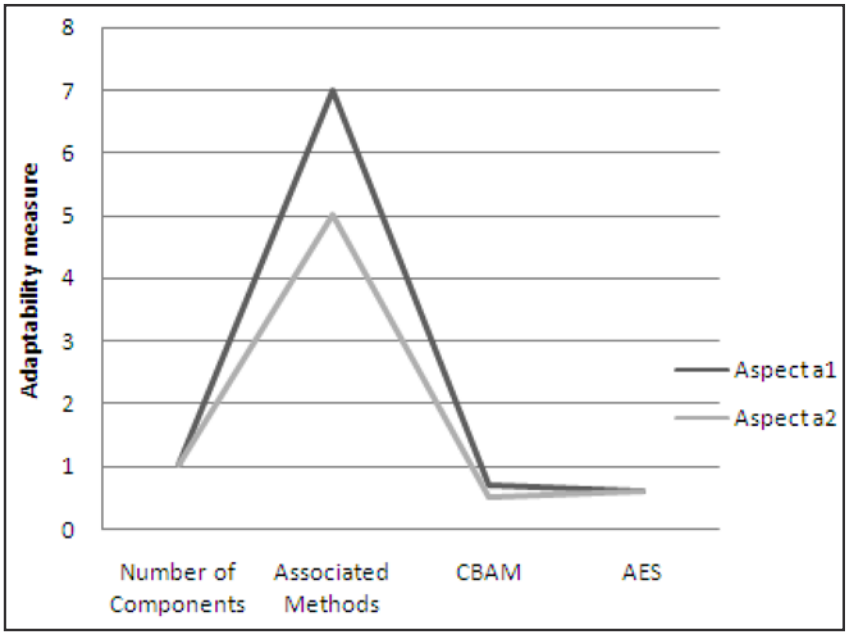

Figure 4. AES Chart

parameters and output parameters respectively. The Remote Interface Reflector component is formally specified using the schema structure as shown in the Table 2. In this structure getInterface is described as a function used to derive the interfaces associated with a component. This function takes component name specified in the set CNAME as an input and produces interface names specified in the set IFNAME as output.

The Remote Method Reflector Schema shown in the Table 3 describes the Remote Method Reflector component. A

6. Architectural Evaluation

The architecture has been evaluated by representing the components using the mathematical structure as shown in Table1 and proved that the dynamic changes can be incorporated without affecting the existing application components. The structure used to formally represent the components of the middleware model as shown in the Table 1 consists of three parts namely (i) 'Schema Name' to specify the name of the middleware component, (ii) 'Domain Declaration' specifies the domain of the property sets of the middleware components and (iii) 'Functional Specification' represents the functions of the components and describes each function with a name and the mapping from an input set to the output set.

Schema Name
Domain Declaration
Function Specification

Table 1. Schema Structure

The sets CNAME, IFNAME, MNAME, INPARAMETER and OUTPARAMETER are declared to represent the names of the component, interface, method, input set PARAMETER is declared to represent INPARAMETER x OUTPARAMETER. Here, interfacename, method-name and in/out-Parameter are declared as the member of the sets IFNAME, MNAME and PARAMETER respectively. getMethod is a function takes interface name as an input and extracts name of the methods and their parameters associated with the interface specified in the input.

Declare a set METHOD to represent (MNAME x PARAMETER) and a set EXECUTE_RESULT to represent (MNAME x OUTPARAMETER). Table 4 describes the

structure of the Remote Method Invoker component. Here Wrapper function is declared as the mapping from METHOD set to EXECUTE_RESULT set. This function executes the method by taking method name and input/ output parameters as input and produces the output as specified In the output parameter of the method.

Remote Interface Reflector
comp-name : CNAME interface-name : IFNAME
getInterface : CNAME ↔ IFNAME

Table 2. Remote Interface Reflector Schema

Remote Method Reflector
Interface-name: IFNAME method-name : MNAME in-Parameter : INPARAMETER out-Parameter : OUTPARAMETER
getMethod : IFNAME ↔ (MNAME x PARAMETER)

Table 3. Remote Method Reflector Schema

Remote Method Invoker
method : METHOD execute : EXECUTE_RESULT
Wrapper : METHOD ↔ EXECUTE_RESULT

Table 4. Remote Method Invoker Schema

The structure of the Aspect Generator component is described as shown in the Table 5. In this component dynRequestMethod is declared as a function to represent point–cut and advice associated with the dynamic requirements. The set DELTACHANGEIMPLEMENTORS, JOINTPOINTS and WRAPPER are declared to specify the code to implement dynamic requirements, joint-points and methods executed using wrapper function defined in the Remote Method Invoker. The dynRequestMethod specifies the function, joint-points and the code to be added in the location specified by the joints-points of the function.

Aspect Generator
code : DELTACHANGEIMPLEMENTORS JP : JOINTPOINTS
dynRequestMethod : (JOINTPOINTS x DELTACHANGEIMPLEMENTORS) ↔ WRAPPER

Table 5. Aspect Generator Schema

The Aspect Weaver class is formally represented using the prescribed schema structure as shown in the Table 6. In this structure dynamicWeaver is the function declared to weave the advice of the dynamic requirements with the wrapper. It takes the advice corresponding to the dynamic requirements and wrapper function as input and generates the function code along with the advice as output.

Aspect Weaver
Wrapper_method : WRAPPER code : DELTACHANGEIMPLEMENTORS
dynamicWeaver : (DELTACHANGEIMPLEMENTORS, WRAPPER) ↔ (DELTACHANGEIMPLEMENTORS x WRAPPER)

Table 6. Aspect Weaver

The fine-grained changes are incorporated into the function of a component without affecting its dependent functions is proved as follows.

Let us consider $m \in MNAME$, δm represents dynamic requirement stated by the user, $wm \in$ Remote Method Invoker. Wrapper, advice$\in$ Aspect Generator. dynRequestMethod –Advice and the adaptation function $f: (m \cup \delta m) \rightarrow (wm \cup advice)$

The function f shows that including dynamic requirements into the method is achieved through integrating the advice with the wrapper method.

$\ni$ a relation $R : M_x \leftrightarrow M_y$ where M_x: *MNAME* and M_y: *MNAME*, dependents of

$M_x \bullet (M_x \cup \delta m) = (wm \cup advice)$ implies that $(M_x + \delta m) <\neq> (M_y + \delta m)$.

7. Conclusion and Future Work

The middleware model proposed in this paper reduces the maintenance effort by making the system to dynamically adapt the changes in the requirements without affecting the existing system structure and behavior. It has been achieved using the principles of (i) separating and defining the dynamic requirements as aspects, (ii) defining wrapper method for dynamically loading the functions on which dynamic requirements to be incorporated using reflection, and (iii) weaving the aspects with the wrapper method associated with the functions of the application components. The principles used to design this model have reduced the effort expended on making the system adaptable to run-time changes in the requirements, which has been proved using mathematical evaluation. The adaptability efficiency of the model has been evaluated using the *CBAM* metric (Conceptual Binding between Aspect and Methods in the component) proposed in this work and proved as an efficient solution.

As future work, it is proposed to integrate the middleware model 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 to 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:* Proc. of the 6th Workshop on Adaptive and Reflective Middleware (ARM 2007). ACM/IFIP/USENIX, November.

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

[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:* Proc. of the 4th International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC 2006). ACM.

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

[5] Cazzola, Walter., Ghoneim, Ahmed., Saake, Gunter (2004). Software Evolution through Dynamic Adaptation of its OO Design. *In*: Objects, Agents and Features: Structuring Mechanisms for Contemporary Software, Lecture Notes in Computer Science 2975, 69-85. Springer, July.

[6] Pemberton, Duncan. RComponents - Reflective & Adaptable Components. http://www.comp.lancs.ac.uk/computingresearch/cseg/projects/REFLEX/REFLEXPaper.pdf .

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

[8] Bencomo, NellyNelly., Blair, Gordon., Grace,Paul (2006). The world is going MAD: Models for Adaptation. *In:* ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, 1900-01-01, Geona, Italy.

[9] Parlavantzas, Nikos., Coulson, Geoff ., Clarke,Mike., Blair, Gordon (2000). Towards a Reflective Component-based Middleware Architecture. *In:* Proc. of the Workshop on Reflection and Metalevel Architectures, June.

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

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

[12] Tarvainen, Pentti (2008). Adaptability Evaluation at Software Architecture Level. The Open Software Engineering Journal, 1 (2) 1-30.

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

[14] Jacquelline FLoch, Svein Hallsteinsen, Erlend Stsv, Frank Eliassen, Ketil Lund and Eli Gjarven (2006). Using Architectural Models for Runtime Adaptability. I*EEE Software,* March, 62-69.

[15] Gail Murphy, Christa Schwanninger (2006). Aspect-Oriented Programming. *IEEE Software*, January, 20-23.

[16] Walter Cassola, Alessandro Marchetto (2008). AOP Hidden Metrics: Separation, Extensibility and Adaptability in SW Measurement. *Journal of Object Technology*, 7(2) 53 – 68.

[17] Przybylek, Adam (2010). An Empirical Assessment of the Impact of Aspect-Oriented Programming on Software Modularity. *In:* Proc. of the International Conference on Evaluation of Novel Approaches to Software Engineering, Athens, Greece, July 22-24, p. 139-148.

[18] Steimann, Adam (2006). The Paradoxical Success of Aspect-Oriented Programming. *In:* Proc. of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Languages, Systems and Applications. *ACM SIGPLAN Notices,* 41 (10) 481 – 497.

[19] Grace, Paul., Lagaisse, Paul., Truven, Eddy., Joosen, Wouter (2008). A Reflective Framework for Fine- Grained Adaptation of Aspect-Oriented Composition. *In:* Proc. of the 7th international symposium on Software Composition. p. 215-230.

[20] Awais Rashid, Thomas Cottenier, Phil Greenwood, Ruzanna Chitchyan (2010). Aspect-Oriented Software Development in Practice: Tales from AOSD-Europe. IEEE Computer, February, 19-26.

[21] Greenwood, Phil., Surajbali, Bholanath.,Geoff., Coulson., Rashid, Awais., Lagaisse, Bert., Truyen, Eddy., Sanen, Frans., Joosen,Wouter (2008). Reference Architecture v3.0. AOSD-Europe-ULANCS-37, 8th January.

[22] Costa, Fabio Moreira (2002). Meta-ORB: A Highly Configurable and Adaptable Reflective Middleware Platform. *In:* Proc. of the 20th Brazilian Symposium on Computer Networks, p. 735-750, Buzios-RJ-Brazil. Brazilian Computer Society.

[23] Cohen, Tal (2007). Applying Aspect-Oriented Software Development to Middleware Frameworks. Doctoral Thesis, Israel Institute of Technology, February.

[24] Wampler, (2007). Aspect-Oriented Design Principles: Lessons from Object-Oriented Design. *In:* Proc. of the Sixth International Conference on Aspect-Oriented Software Development., Vancouver, British Columbia, March.