2003-04-27

An Environment For Specifying and Executing Adaptable Software Components

Sudeep Prabhakar Unhale
Worcester Polytechnic Institute

AN ENVIRONMENT FOR SPECIFYING AND EXECUTING ADAPTABLE SOFTWARE COMPONENTS.

by

Sudeep Prabhakar Unhale

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

May 2003

APPROVED:

Professor George T. Heineman, Thesis Advisor __________________

Professor Lee A. Becker, Reader __________________

Professor Micha Hofri, Department Head __________________

Abstract

One of the difficulties of Component Based Software Engineering (CBSE) [1] in reusing pre-existing components is the need to *adapt* these components to work within the desired target systems [2, 3, 4, 5, 6]. Third-party or in-house Commercial Off-the-shelf (COTS) components may not always have the required exact functionality demanded by the builders of the target system, so these systems have to be either modified [2, 3, 4, 5, 6] or adapted to provide this required functionality. Modifying these components may not be always practically possible.

In this thesis, we propose an infrastructure that supports the active interface adaptation technique [3, 8, 9, 10]. This infrastructure directly addresses the problem of effectively packaging components for third-party use, adaptation, and deployment. Doing so we support both component designers and third party application builders. Further we evaluate our approach using several adaptations over the case studies.

TABLE OF CONTENTS

LIST OF FIGURES

Number *Page*

LIST OF TABLE

ACKNOWLEDGMENTS

This work was based on the National Science Fund under grant number CCR-9733660.

I want to express my sincerest appreciation and thank Professor George T. Heineman, my thesis advisor, for his patience, advice, guidance and continual support.

I also want to thank Professor Lee Becker for being a reader for this thesis.

Special thanks to, Chunling Ma for the CSLProfiler tool and Paul Calnan for the AIDE tool.

I also wish to express sincere appreciation to all friends and colleagues for being there and for the support.

Last but not least I would like to thank my parents and sister for making all this possible for me and for their infinite love, support and encouragement

I dedicate this work to my parents.

C h a p t e r   1

INTRODUCTION

1.1 INTRODUCTION TO SOFTWARE COMPONENTS

1.1.1 Motivation for adapting Components

When building large-scale software systems using Component Based Software Engineering (CBSE), some functionality desired in the new system can be found in pre- developed components, either in-house pre-existing components or third-party commercial Off-the-shelf (COTS). Thus we have a trend of an increasing use of external components by new application builders. This demand implies improvements in how components are designed, documented, assembled, adapted and deployed.

Heineman argues that a component market place will exist when application builders can adapt software components to work within their applications [3, 5, 6]. The motivation for doing so is that reusing a component avoids implementing same functionality from scratch and minimizes maintenance costs [3].

1.1.2 Software Components and their types

A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard [1].

Components can be deployed independently and can be given run-time resources. Components can independently interact with other components to form a software system. These systems can be further used as component themselves interacting with other sub-systems to form a large application/system.

Components can be of two types: black box components and white box components. White box components enable the application builder code to access the source code of the component. Components in most cases are of black box types provided by third party developers where source code is not available. The black box property of components is as essential as abstraction and information hiding are to Object Oriented programs [2].

The component model defines how the components interact with each other. Large complex systems are being constructed on the basis of these component-based models. Component technology leads to a strong commercial market for pre-developed software components.

1.2 ADAPTATION OF PREDEVELOPED SOFTWARE COMPONENTS

1.2.1 Introduction

Pre-developed software components are increasingly gaining importance as they can be assembled, reused and deployed to form larger systems. Component vendors develop these components in a developer environment and application builders deploy them in the customer environment. However, it is observed in most cases that the requirements specified by the application builders to assemble a larger system using third-party components may not exactly match the requirements of the application. To overcome this difference, one must adapt the component for the additional functionality or to remove an undesired functionality.

This adaptation can be done in two ways: either by modifying its source code (white-box) or by wrapping it (a blackbox technique).

1.2.2 Component Adaptation

If an application builder decides to adapt the component, he must understand the complex behavior and functionality of its code. Any change in either of them may break the structure of the system specified by component designer, which has to be avoided. While component adaptation occurs during integration, it is different from component customization. In component customization, the application builder tailors the component using certain pre-defined options. It may not always be possible to satisfy the additional requirements.

Prior research identifies some requirements for component adaptation [4]

- The adapted component may behave differently but should be used in the same way as the original component would have been used.

- There should not be any additional effort in making the adapted component integrate with the target system and the client-side view should be maintained. This property of adaptation is called *transparency*.

- The original component should not have any knowledge of the adaptation. The component must be open to future adaptations and should not lose its identity as an original component.

Software engineering phases have to be considered with respect to component adaptation [16]:

- *Design phase*: The component designer has to specify the interfaces, interoperability and relationships between components.

- *Implementation phase*: The component is constructed from the design specifications. Writing the code, obtaining all the dependable components (can be COTS), implementations of interfaces, data-structures, database tables, compilation and linking of the source files.

- *Deployment phase*: The component is to be deployed into a component infrastructure. The composition of the components should also be considered during adaptation.

To adapt a composite component [7], the outer component might have to be modified or adapted. During composite component adaptation, original component's out interface should not change in terms of declaration or the prototype. Similarly the original component's provided interfaces should remain unchanged.

1.3 THESIS OUTLINE

The outline for this theses goes like follows,

1.3.1 Further in this chapter, we cover the assumptions and contributions of this thesis. Also some popular white box and black box component adaptation techniques are discussed with emphasis on the Active Interfaces [3, 19].

1.3.2 Chapter two covers the need for Component Specification and explains the Component Specification Language [3, 5, 6], and its parsing structure.

1.3.3 Chapter three discusses about the case studies done and the adaptations that were proposed for the same. It also discusses some of the case studies with respect to different component deployment strategies like CMI [1], EJB [20].

1.3.4 Chapter four discusses in detail about the design methodology and describes the component adaptation mechanism we are proposing. It also discusses the various aspects of the approach we are suggesting like the supporting tools, extra adaptation code and deployment strategies.

1.3.5 Chapter five discusses the evaluation of our proposed approach on the basis of the practical implementation of the adaptation and also on the basis of integration of the component adaptation technology with the popular component deployment techniques.

1.3.6 Finally Chapter six provides the conclusion for this thesis and presents a vision for the future work towards the concerned area of component adaptation.

1.4 ASSUMPTIONS:

This thesis is based on few assumptions with respect to responsibilities of actors as follows

1.4.1 Responsibility of component designers

Component designers are responsible for design, construction and packaging of a software component. They would also be responsible for delivering an adaptable component. They may provide necessary information that aids the application assembler [19] to have this component used as required for the target application. As far as the adaptable component packaging is concerned if the component designer is packaging the component giving the source level access to the application assembler, an application assembler or a component adapter can be responsible for converting this component into its adaptable form using the tools like AIDE [10], used to instrument the components code as required or any other technology to obtain similar result.

1.4.2 Responsibility of application assemblers

Application assemblers are responsible for assembling the target software system from the component vended to them by the application designers. They also have responsibility to know details about the component and how to integrate it with the new system. They may have some documentation from the concerned component designer/vendor or they can rely on the tools like CSL-Profiler [15], which gives information about the given black box component in the form of a CSL specification.

Application assemblers may have as deep knowledge about the component as application designer, which is mostly the case when the component vended to them is a white box component. In a practical scenario, an application assembler knows somewhat about the component and the way it can be deployed.

1.4.3 Responsibility of component adapters

Component adapters' responsibility is to convert the component vended into an adaptable component so as when it is reused by a component assembler it can be adapted accordingly for the new system to be built.

Component adapters can do this using a tool like AIDE [10], which is used to instrument the vended components code as required thus making the component an adaptable one. One can also attempt to get an adaptable component by manually changing the source code or instrumenting the source code using other techniques. Most of the approaches require source code level access to the component, which supposedly passed on by the component designer.

Typically a component adapter can also be a system assembler who enjoys the component adapter level access to the component vended.

1.5 CONTRIBUTIONS:

This thesis makes the following contributions

1.5.1 CSL parser:

This intelligent JavaCC based parser is useful for parsing the CSL scripts and populates the CSL container infrastructure. This parser is developed to act as an individual component and thus can be reused for any other research or purpose.

Also an independent hierarchical object oriented structure is created to store the information obtained from the CSL scripts to enable efficient use of information provided about the component, its adaptability and the actual location of code of the adaptation.

1.5.2 CSL container infrastructure

The CSL container infrastructure enables the practical adaptation of a black box component and its deployment.

In this thesis, we have successfully presented the way this container infrastructure works and how to use the environment to adapt a given component and integrate it into a new system. This infrastructure can be used as a template to attempt future component adaptation and deployment.

1.5.3 Prototype adaptations

We also present several case studies on using Active interfaces to adapt the given application for desired adaptations.

We show in detail how to adapt our application for the basic three adaptations and also further for evaluation purpose we have contributed by selecting and attempting four more adaptations, all of which can be seen as effective examples or templates to carry on future adaptations of any blackbox component using the technique we have proposed in this thesis.

1.5.4 CMI integration of CSL container

We also set out to integrate adaptation technique with component models. Component Model Implementation [1], like EJB [20] and CORBA [21], is a component-based architecture to deploy components. Using CMI we can have multiple components whose life cycle is controlled during its run-time by a container.

This component deployment technique was used as a base for validating the deployed adaptations as presented in this thesis. We integrate CMI with the CSL structure to enable deployment of the adapted components in the CMI environment.

1.5.5 Simple extended CMI tutorial

This thesis also provides an extended simple tutorial for deployment of the adaptable components and their adaptations using the method provided described in this thesis in the given CMI environment.

This tutorial can be seen as the steps to use towards importing the component adaptation technology in the component deployment policies supporting such components.

1.6 BACKGROUND

There are various ways in which we can adapt software components. In this thesis, we have focused on several related white box and some black box adaptation techniques including Binary Component Adaptation [17], Active Interfaces [3,19], Superimposition [7], Component Adaptors [18], Wrapping and Inheritance.

1.6.1 White box adaptation techniques

White box component adaptation techniques require code level access to the component designer or application assembler. The simplest white adaptation technique is to change the part of the source code for desired adaptation. This is the most straightforward approach. It needs unrestricted access to the code. Another similar white box adaptation technique is “Copy-Paste” method. Here an application assembler may copy paste the required code for adaptation from the library or other source and get the desired adaptation done. This method can be effective at times but results in overheads of having multiple copies of the code. This is one of the popular white box adaptation techniques.

There is one more adaptation technique, Inheritance, which can be classified in both white box and black box category depending on the technology (language) it is being implemented in. This approach also requires the software assembler to have detailed information regarding the working of the component super class he is inheriting. This approach also depends on the kind of language it is implemented in, first the language should support inheritance and there can be some restrictions in doing so .For example, the private, protected types in java.

1.6.2 Black box adaptation techniques

We do not have code level access in blackbox based adaptation approach. One of the popular blackbox adaptation approaches is “Wrapping”. The component to be adapted is embedded in a new component whose interface matches to that of the application requirement. Wrapping allows components to be composable or be aggregated to compose new functionalities. Wrapping provided separate location for the adaptation code. Wrapping may not be always a good idea since it restricts reuse and may end up with complex excessive adaptation code. Wrappers are typically useful while integrating components using different technologies [9].

There is one more alternative to wrapping that is by using a proxy component. One can generate the same adaptation effect as wrapping but adaptation code is written in separate component, which acts like a filter. This technique is better than wrapping with respect to reuse factor but does not provide transparency as the client deals with the proxy instead of actual component. Component Adaptors [18] are an extension to proxy adaptation technique. They can be generated automatically and use high-level description language for specification.

Binary component Adaptation [17] is another black box adaptation technique where the components to be adapted by the component assembler when they are loaded into memory for first time. BCA manipulates the components in their compiled form hence require no source code. Transparency is maintained and flexible adaptations are allowed. BCA also leads to some load time overheads and is very language specific. It can be configured using delta file.

Superimposition technique [7] superimposes behavior to extend software components. It is based on principle that a component and its adapting functionality are different entities, which are integrated appropriately. It requires components to have interface and adaptation of a component is supported by ability to change the mapping from interface to the concerned methods. There can be different types of adaptation behavior predefined for some specific components. This technique is composable and configurable with a good reusability factor.

Note: *Detailed comparison between adaptation techniques can be found in references [4, 7, 9, 23]*

1.6.3 Active Interfaces

Active Interfaces [3,19] mechanism inserts additional interfaces that are invoked in components with the already exposed interfaces. Callbacks are inserted for members of the component to be adapted. The members being referred here are methods for the given component. There are hooks that map the pre and post condition of a given method explained more in Section 1.7. Active Interfaces are language Independent. The components to be adapted have to implement an *Adaptable* interface, which allows setting a component adapter for the given component. Methods when adapted have an associated component adaptor that controls the insertion of hooks in those methods. All additional adaptation code is written in glue code, which is external to the component as in case of the proxy adaptation.

A typical callback specifies the adapted method name, the phase (before, after or negotiate), the glue object and new method with its properties (parameters). Here the new method is implemented in glue code. The whole adaptation cod lies in new method. The callback used for monitoring the events on basis of the pre and post conditions. Typically information sent to monitor can be extensively evaluated on the pre-phase of callback and accordingly corresponding denial, augmentation and overriding of the functionality takes place. Functionality is denied if the service provided by a method is refused. Functionality is augmented when values of parameter are altered and functionality is overridden when the concern method is overridden. There is an Active Interface Development Environment (AIDE) [10], which addresses towards developing components with Active Interfaces. AIDE had a tool that instruments the given method of a component or all method and inserts the right hooks for imposing the pre and post conditions monitor.

Active Interfaces offer many advantages

- Separation of adaptation code from actual component to be adapted.
- Callbacks can be inserted for all methods irrespective of whether they are internal or exposed.
- Language independence.
- Runtime support for dynamically adding or removing hooks.
- Original component even though inserted with callbacks same as it used to before applying Active Interfaces.

There are few disadvantages

- Current tools like AIDE currently only support Java.
- Active Interface mechanism requires Source code to instrument the methods

In this thesis, we have developed an infrastructure that supports adapting blackbox software components using an approach developed by Heineman [2, 3, 4, 5, 6]. Also we have focused on enabling component designers to build an adaptable component for which we proposed a framework, that packages and deploys these adaptable components for *third party use* and *adaptation*. A Component Specification Language is to be used as a part of the infrastructure for providing the semantic detailed information about the adaptable component and how it is adapted. In earlier papers [3, 5, 6], Heineman provided some examples of use of CSL; we have designed and implemented a parser for CSL and an infrastructure that accepts and processes CSL specifications.

1.7 ADAPT PROJECT

Previous research [3, 4, 5, 6] has identified relevant concepts with respect to component adaptation and building of adaptable components. The ADAPT project sponsored by an NSF grant, involved building groundwork and foundation for this thesis, which will conclude this research project.

Previous research proposed the concept of an *active interface* (Section 1.6.3) where, an active interface decides whether to take action when a method is called, an event is announced or a protocol executes [3].

There are two phases to all interface requests

- *Before phase*: occurs before component performs any steps towards executing the request.
- *After phase* : occurs when component has performed the execution of the request.

Thus we have a standard way of altering the behavior of a component by interposing an entity to intercept messages and/or events as shown in figure below.

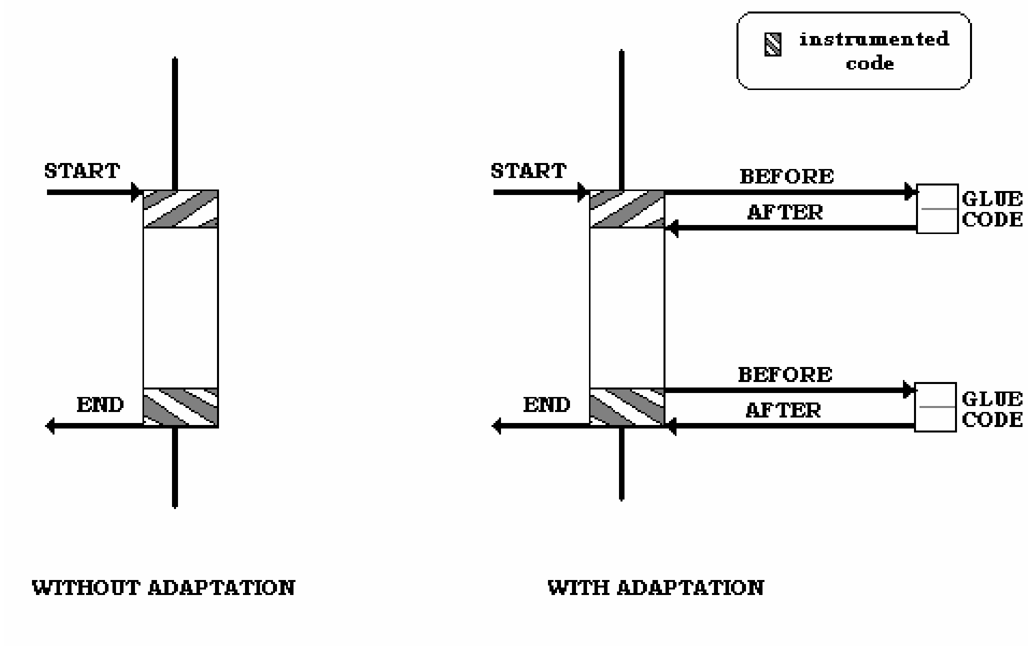


Figure 1.1: Active Interface.

The first case in Figure 1.1 typically reflects working of a blackbox component without adaptation whereas the second case reflects the adaptation of the same component done using before and after callbacks.

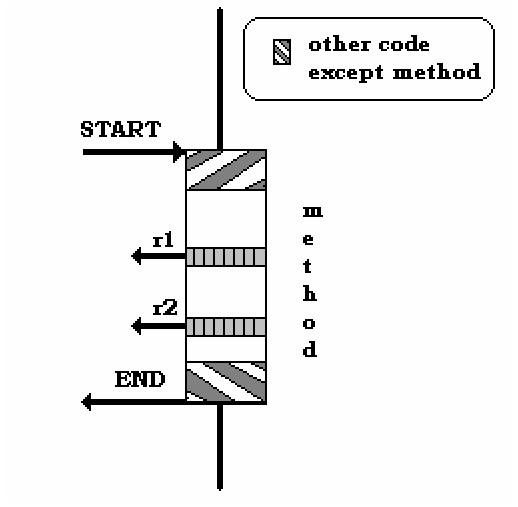


Figure 1.2: Multiple returns from component.

Figure 1.2 typically reflects that for a given method in a component, there may be multiple return points such as r1 and r2 in the above figure. These types of methods are also candidates for adaptation.

The groundwork for the ADAPT project was laid in the form of published research work and some implementations such as Component Adapters, active interfaces, and the example Spreadsheet application [see Appendix B], which is our motivating example. Some contributions to this research have also come from some previous theses affiliated to this ADAPT project, where some of the concepts were used and implemented; most notably, DASADA [22].

DASADA uses AIDE (Active Interface Development Environment) [10] to insert (wrap) software probes into a software component. These probes are used to report system events for the concerned component. The probes also provide the entry points to the concerned component/system, wherein system events propagate across “active connectors”. These events can be used further for various uses like gauging the application state, or transforming that applications state for using the current event or set of events. These event led gauges can further lead to automated decisions to be taken by the main DASADA system and also for diagnostic purposes.

Most of other work in this field concentrates on either a single domain and /or support the as-is reuse of a given component sometimes revealing internal implementation of the component or sometimes even modifying the original implementation as in case of white box component adaptation. Our research strongly supports the reuse and adaptation of a black box component on the assumption that this component is made adaptable. We attack this problem by presenting a template-like framework that would motivate a component designer in making adaptable component or reengineer the existing piece of code/component to make it adaptable.

1.8 SUMMARY OF CHAPTER

In this chapter we go through basics of predefined component adaptation and cover the various software engineering phases to be considered for the same in Section 1.2. Further in Section 1.3 we discuss the general outline for this thesis brief giving overview of each and every chapter.

In section 1.4 we cover briefly the assumptions with respect to the actors involved in the adaptation process. Section 1.5 discusses the contributions done in this thesis. Section 1.6 talks about different component adaptation techniques. Section 1.6.3 and Section 1.7 explain Active Interface approach in detail and also discusses about the ADAPT project.

C h a p t e r   2

COMPONENT SPECIFICATION

2.1 INTRODUCTION

We need a language to specify semantic information about the component and the adaptations that are to be made. This specification language must be easily understood by the application builder and easy to specify by the component designer. We propose the Component Specification Language, which was presented only by example in previous research [3,6].

The need for a specification language here pays an important role for the reason we propose a new language over using an existing one. There are three basic kinds of Specification Languages with our perspective,

- Interface specification language (ISL)
- Component Specification Language (CSL)
- Architecture Specification Language (ASL)

These languages cater to different needs and have special purposes. ISL or Interface Definition Language (IDL) [28, 25], is primararily used to specify the interfaces of the given system, / component. The CSL [26, 27] is used to specify about the components and is mainly used by technical architects to express system platform, distribution, and other non-functional requirements to assist system packaging. An ASL is used to describe about the software architecture of the given component. ISL confines itself to specification of the interface the component and its information but has no way of suggesting any behavioral aspects of the component. CSL or CDL (Component Definition Languages) does to an extent is successful in describing the component and its interface but has no provision to specify where the extra code (glue code) for the adaptation lies. Some researchers [27] do have special GSL (Glue Specification Language) to solve this purpose. ASL or ADL (Architecture Definition Language) does have some provisions that make them candidates for use but there are too many specification details, which may not be needed for our specification purpose. So we strongly feel a need to propose a language, which allow specification of interface and other details about the component like (ISL and CSL integrated) and also allows one to specify the Glue code. Some other researchers have taken similar language based decisions as in case of [24,27].

We are not modifying the code nor are we doing software customization because the customization mechanisms will be built into adaptable software components. To illustrate our approach, we will use as our motivating example, a spreadsheet application, which conforms to the JavaBean architecture [20]. This application is a good candidate to be adapted and can be treated as a blackbox component. We have already put in hooks for adaptations as required by the Active Interface mechanism [3,19], which were manually inserted in form of supporting code and we can thus consider this candidate component as an adaptable component.

CSL will also be used to specify the overall architecture of the application, and one goal of the research would be to practically determine which architectural styles are best suited for CBSE. The framework is proposed to work as shown in figure 2.1.

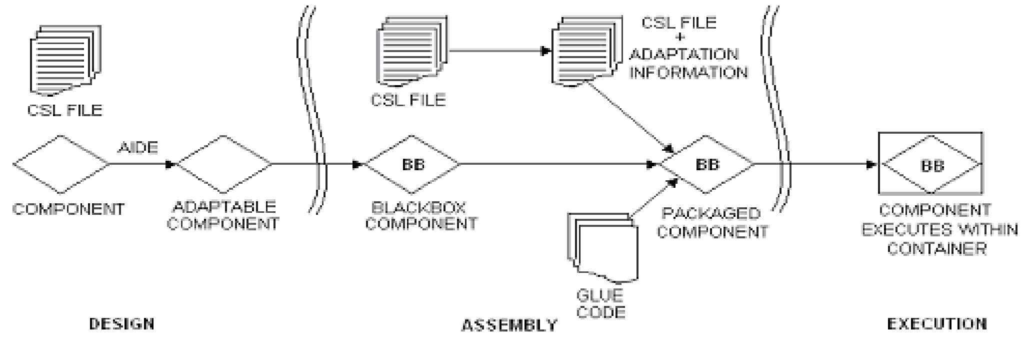


Figure 2.1: Component Adaptation process.

We need adaptable components for our evaluations. We will incorporate into our project AIDE, a technology to upgrade a software component to support Active Interfaces [10]. This technology helps us to identify the adaptations allowed by the given component using a customized tool. Thus the designers would now be able to select the methods in the component they want to be adaptable and the concerned before-after callbacks would be installed. There is also a tool, CSL-profiler [15] that will use the information from the AIDE tool (See tools section in Chapter 4) and generate the concerned CSL specification for that component or application describing the adaptations available to the application builders and/or concerned third parties.

In Figure 2.1, the component designer creates an adaptable component- using AIDE and provides CSL specification of the Adaptable properties (see Appendix). Then Application builder adapts the component by augmenting the CSL specification to define the before/after callbacks that are used and providing relevant glue code .The final packaged component will execute within container.

We know that some components are poorly documented and/or very little is known about them so if one has to adapt such components we need a mechanism to get semantic information about the components from a CSL file. We can also use the above technology to obtain information about such components using CSL specifications. Our framework launches adaptable components within a given container. This container is also responsible for parsing the CSL Specification and installing the appropriate glue code methods using the Adaptable interface created by AIDE .The container will also support component arbitrator as described in [3]

2.2 CSL INTRODUCTION

A CSL file consists of elements describing the component or application and it consists of different sections. For a component CSL we have the information about its public /private /protected methods. Also we have information if any of these methods are adaptable. From the component CSL we can also determine if this component is adapting another component and if so, where the extra information about the adaptation code (glue code) is to be found. Additional information about the methods of this glue code can be specified as action statements and information about the before /after callbacks are provided in the CSL file suggesting the adaptation. A component can also have properties that are achieved by get /set methods. More on this is discussed in the component CSL example.

As far as an application CSL is concerned it has information about the interfaces it implements. This CSL also consists of the number of components associated with the application CSL. This CSL also provides information about its public /private /protected methods if any. An application can also have properties that specify the get /set features for that application.

2.3 CSL GRAMMAR

A CSL file starts with a description of an application or/with a description of a component.

2.3.1 Component CSL

A Component CSL, which also is a part of an application, consists of

- The name of the type the component is an instance of or (the one that the component is adapting).
- The name of the Class /code where the new adaptation code/logic is written
- The properties associated with the component
- The methods associated with the component.
- The methods that can be specified with before /after clause and /or negotiation clause and their descriptions.
- The public, private and protected methods associated with the component.

A component CSL grammar is similar to that of a component CSL grammar found in the component section of the Application CSL. The reason we have two different types of CSL specifications is to enable the reuse of a CSL specification for a required component.

A typical Component CSL file can be shown as follows

```
component  <name> adapts <component >{

        Class
        Actions …
        Properties …
        Methods …
        Public/Private/Protected  methods…
}
```

<u>Note</u>: *A Component CSL can adapt a previously adapted component*

Sample of Component CSL 1

```
Component spreadsheet.Spreadsheet{
//sample specification of  Spreadsheet component

Public {
   void setFunction (String,Function);
   Function getFunction (String);
   void clearFunction ();
   void installFunctions ();
   String getValue (String);
   void setValue (String,String);
   void setValue ();
   float getNumericValue (String);
   float calculateFunction (Expression,String,Enumeration,Node);
   void handleSpreadsheetEvent (SpreadsheetEventObject);
   void addSpreadsheetListener (SpreadsheetListener);
   void removeSpreadsheetListener (SpreadsheetListener);
};

Protected {
   adaptable float evaluateConstant (String);
   adaptable void evaluate (Node);
};

Private {
   void setDebug (boolean);
   final boolean getDebug ();

};

};
```

2.3.1.1 Component CSL grammar

The component CSL can be described as adaptation or instance or just the component itself. If a Component adapts some application more details of the component can be specified as shown in sample of CSL 2 (shown further in this Chapter).

This component description is made of a couple of statements followed by the keywords like class, action, property, method, before, after, negotiate so on.

class is the name of the Class where the additional functionality is placed. For example in our case study “Glue” Class. This class clause is typically used to indicate the external code/class file that is written with the new adaptation functionality associated to this component.

property is a brief description of the get-set data-members of the specified component . In the component clause of the CSL description where the word following keyword “adapts” is the name of this type being adapted.

action is brief description of the *method* of the Class/file specified in the “class” clause of the CSL description, where the word following keyword action is the name of this method and is also termed as action name .The attributes of the method are given here in form of parameters, which are usually a fully quantified name. The parameter generally represent the input datatypes of the method in the given class specified for this component adaptation/description. The parameter type precedes the parameter names.

action are typically used to specify the methods associated with the external code that is written with the new adaptation functionality indicated by the *class clause*.

method Indicates brief description of the method of the class/module specified component is adapting. In the component clause of the CSL description where the word following keyword “adapts” is the name of this module being adapted. Word following keyword “method” is the return data-type of the Method specified and next word following is the name of this Method. The attributes of this method are given here in form of parameters, which are usually a fully quantified name. The parameter generally represents the input parameters of the method specified. The parameter data-type precedes the parameter name.

Following the method parameter description there are open curly braces where in lies the other concerned description, generally about the adaptations information with respect to this specified method. This adaptation information is basically based on three keywords, *before, after* and *negotiate*. In *before/after* clause the action is specified using either of the keywords specifying if the action is to be invoked before the specified method or after the specified method. In the above example for evaluate method, we specify in such a way that the script suggests to invoke beforeEvaluate action before the method evaluate and on similar lines the script suggest to invoke afterEvaluate action after the method evaluate in CSL example 2.The actions here are presumed to have the required functionalities to carry out the required adaptation.

before & after keywords are typically used to specify and suggest how the adaptations that are required are done using action (atleast specifying to invoke that action) before or after the invocation of the specified method.

In the *negotiate* clause, the action is specified with the parameter type associated with the Method specified e.g. *refreshList* in the second line of CSL example 2 .A unique negotiate name is assigned which follows the keyword “negotiate” which is also termed as negotiate policy name.

Using this negotiate clause one can suggest adaptations such that every time the specified method is invoked the suggested adaptation action filtercells takes the concerned parameter input and processes or updates it for some required functionality. The negotiate keyword is typically used to specify and suggest how the adaptations that are required are done using action using the input parameter of the specified method.

Note: *Methods are essential and an important part of the component CSL from the adaptation point of view one should be thus very careful while writing this portion of CSL since a slight mistake here may result in a logical error which can be hard to debug.*

Brief description of the public/protected/private methods may be specified in the component about which CSL description is, where the area following keyword public/private/protected is the area where the concerned methods declared are described for the given component and they are termed as public/private/protected methods/listeners .the methods and listeners are listed in between the curly braces. As in CSL example 1.

adaptable keyword before a method indicates brief description of the method/component specifying whether or not that particular method or component is further adaptable. The first word of a line of the CSL description here has to be "adaptable" implying that the concerned method/module is adaptable. It makes it easy to know whether the component and/or the method is useful for adaptation in future.

final keyword before method indicates brief description of the method/component specifying whether or not that particular method or component is further adaptable. The first word of a line of the CSL description here has to be “final” implying that the concerned method/module is non-adaptable further. It makes it easy to know whether the component and/or the method is useful for adaptation in the future.

NOTE: *Final & Adaptable keywords are mutually contradictory.*

2.3.2 Application CSL

An Application is an assembly of software components; for example, a spreadsheet application that contains information such as

- The interfaces implemented by that application
- The components associated with the application
- The properties of the application
- The public, private and protected methods associated with the application

This application can be described in a way that reflects the kind of adaptations that are supported by the applications as well as the adaptations that are performed on the components of this application

A typical Application CSL file is shown as follows

```
application <name> {
        Interfaces …
        Properties…
        Components…
       Public/Private/Protected methods …
}
```

A Typical sample of an Application CSL is

```
application FinalApp {

// application CSL for spreadsheet application FinalApp
implements      ActionListener ,      TableListener ,
                SpreadSheetListener , TextBeanListener ,
                Serializable ;

component vs instanceof ScrollbarBean;
component hs instanceof ScrollbarBean;
component newcomponent instanceof adaptme;

component tb    instanceof TableBean;
component tbBox instanceof TableBean;



property int viewHeight;
property int viewWidth;
property CellRegion tableSelected;
property CellRegion tableRSelected;


public {

// public methods

void init();
void init_gui();
void init_components();

// various Listener interface
void actionPerformed(ActionEvent);
void handleTableEvent(TableEventObject);
void handleSpreadsheetEvent( SpreadsheetEventObject);
};

private {

};

protected{
};

}
//end of application CSL script
```

2.3.2.1 Application CSL grammar

Application CSL Starts with a keyword "*application*". It indicates that the application named "FinalApp" is being described in the form of CSL. If an application implements some interfaces they are specified in the "implements" clause of the CSL file as shown in example of an application CSL. Further the application CSL can have some components associated with it. These components described can be adaptation or instance.

Note: *Here the rest of the grammar included in the Component adaptation is the same as described in the above component CSL grammar description.*

Components are typically used to specify and suggest the required adaptations associated with an external code that is written with the new adaptation functionality indicated by the *class clause* in the component. (incase of "adapts" keyword.). Components are also typically used to specify application module associated with a different module that is not in the component domain. (Incase of "instanceof","extends" keyword.)

Property indicates brief description of the data-member of the specified application. The word following keyword property is the data-type of the

data member specified and next word following is the name of this property. There are no more attributes to this property. Properties are accessed through get and set methods. In general, Java strongly encourages class designers to use getProperty() and setProperty() as the name of the methods for accessing Properties. Properties are typically used to specify the variables/data members in the application.
There also brief descriptions of the public/protected/private methods specified in the application about which the CSL description is, where the area following keyword public/private/protected is the area where the concerned methods declared are described for the given application and they are termed as public/private/protected methods/listeners.

2.3.3 APPLICATION ASSEMBLERS ADAPTATIONS

Application assembler is the one who tries to get the information of the component and needs to adapt this component to be a part of the new target system. The assembler writes the adaptations for the concerned component in a separate glue code and has to modify the CSL scripts. The assembler also writes few new CSL scripts suggesting the adaptations to be done.

The component Specification example 2 is the sample of one of the CSL scripts suggesting adaptation for our case study, *Spreadsheet application*.

Sample of a Component CSL 2

```
component adaptme adapts spreadsheet.Spreadsheet{

    class Glue;

    action beforeEvaluate (adapt.spreadsheet.Node node );
    action afterEvaluate (adapt.spreadsheet.Node node );
    action beforeEvaluateC (java.lang.String s );
    action beforeEvaluateMail (java.lang.String s );
    action filterCells(java.lang.Vector refreshList);


    method void generateRefreshEvents(java.lang.Vector
                                         refreshList) {

             negotiate RefreshPolicy:
             filterCells(refreshList);
   };

   method void evaluate (adapt.spreadsheet.Node node){
             before beforeEvaluate (node);
             after  afterEvaluate (node);
    };

   method void setValue (java.lang.String dest,java.lang.String
                                                        value){
             before beforeInsert(dest,value);
    };

   method float evaluateConstant (java.lang.String s){
             before beforeEvaluateC (s);
             before beforeEvaluateMail(s);
   };

   };//end of component
```

The *class* keyword suggests that the extra code required for the adaptations, is in class called Glue. The *action* declarations give more details on the methods of this Glue class. For each method given by *method* keyword, we have some special information that gives us the actual adaptation hierarchy using the three keywords *before*, *after* and *negotiate.* When used appropriately we can have the adaptations to be reflected in the CSL script as in component CSL example 2.

For example here we write

```
method float evaluateConstant (java.lang.String s){
            before beforeEvaluateC (s);
};
```

Where we want to check if the string inputted is a URL, if so we want to do something with that. So here we suggest that before the method evaluateConstant() we will invoke the method beforeEvaluateC() which is in *glue code* and handles the required adaptation .

Similarly we can write for other adaptations as shown in component CSL example 2

2.4 COMPONENT SPECIFICATION LANGUAGE PARSER

As proposed in this, the Component Specification Language is implemented using a standard JavaCC platform (Java Compiler Compiler Technology).The proposed grammar was implemented to follow the rules as given in the appendix C.

Typically input to a CSL parser file is a csl file, which is written by application builder in a manner following the rules as suggested by the CSL-parser, reads the csl file, parses it and this information can further be used for computation. Thus this gives CSL flexibility of being used for the project purposes as in project ADAPT ,we have tried to use the CSL Language as a medium to express the information about the component as well as the information regarding the adaptations done with respect to the component. The process in all is same as Component Adaptation mechanism, discussed earlier.

CSL parser generates appropriate error messages when the input CSL file doesn’t conform to the grammar specified. Also further the CSL parsing can be done more intelligently so that the parser checks the availability and validity of the information from the CSL before it passes the whole information for processing to the next phase.

Just having information is not enough but one has to use and store that information in a systematic fashion. For the ADAPT project we decided to have an infrastructure of objects such that we can systematically store the information from a given CSL file as well as process it and for that matter regenerate it. If the CSL file is for an application an Object of type CSLApplication is generated consisting of the information from the CSL file else if the CSL file is for an component an Object of type CSLComponent is generated consisting of the information from the CSL file.

Further these main classes CSLApplication and CSLComponent deal with different classes so as to make a complex hierarchical object of the CSL information according to the input.

The distinct advantage of having this kind of arrangement is that one can reuse the information provided in a CSL script for multiple times during runtime without having to parsing the CSL file again (unless specified). We can regenerate the CSL file output from the objects formed previously by parsing the CSL script file. Doing so can be beneficial cause the csl then generated may contain all the CSL specification of the referenced members in the main application CSL script and so on.

2.5 CSL PARSER DATA STRUCTURE

The Component Specification Language grammar is annotated with actions to construct data-structures representing the information shared within the CSL script file. We have implemented these data-structures in Java, whose structure is as shown in the class diagram, Figure 2.2.

This structure is a unique structure made of different component classes placed in hierarchy accordingly to CSL grammar literal hierarchy and hence is a totally compatible structure to store the information regarding the CSL script parsed.

The Application CSL details are stored in a CSLApplication class object which consists of different subclass objects like interface class objects, CSLComponent class objects, property class objects and Public class objects.

- Component CSL details are stored in a CSLComponent class object that consists of different subclass objects like CSLAdaptation class objects, property class objects, and Public class objects.
- CSLAdaptation class object consists of different subclass objects like Action class objects, Method class objects, and Parameter class objects.

- Method class object consists of different subclass objects like Neg class objects, BeforeAfter class objects, and Parameter class objects.
- Neg class object consists of different subclass objects like negMethod class objects.
- Public class object consists of different subclass objects like Method class objects.
- Action class object consists of different subclass objects like Parameter class objects.

All these objects in combinations represent a single object for input Application or Component CSL script information during the run time.

2.6 SUMMARY OF CHAPTER

In this chapter we establish the need to specify components and to propose the new Component Specification Language. Section 2.1 and 2.2 gives a brief introduction about the CSL.

We further in Section 2.3 give the basic grammar for the component Specification Language. In Section 2.4 and 2.5 we discuss about the CSL parser and the data structures we require to hold the parsed CSL information.

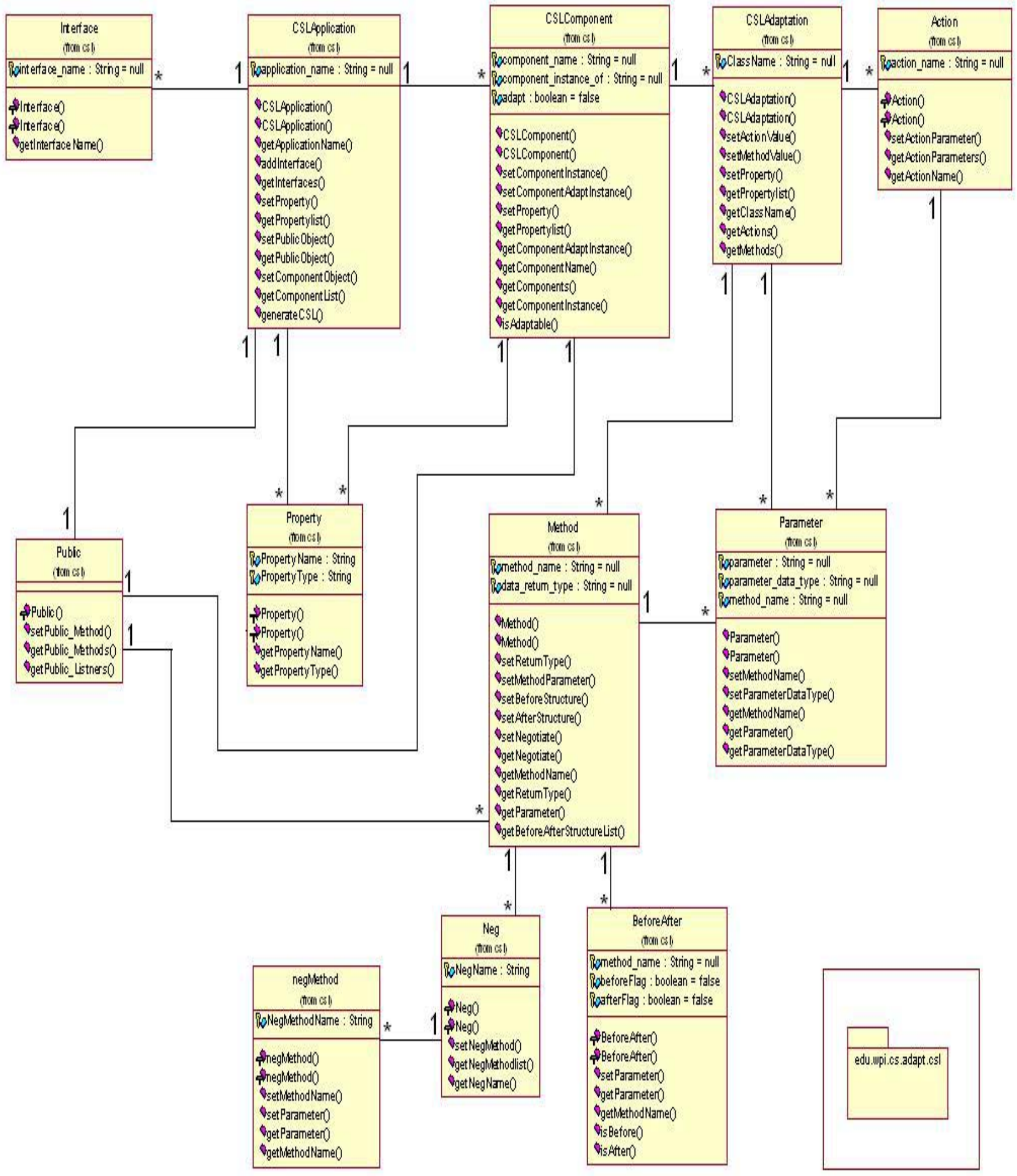


Figure 2.2: CSL Parser Data Structure.

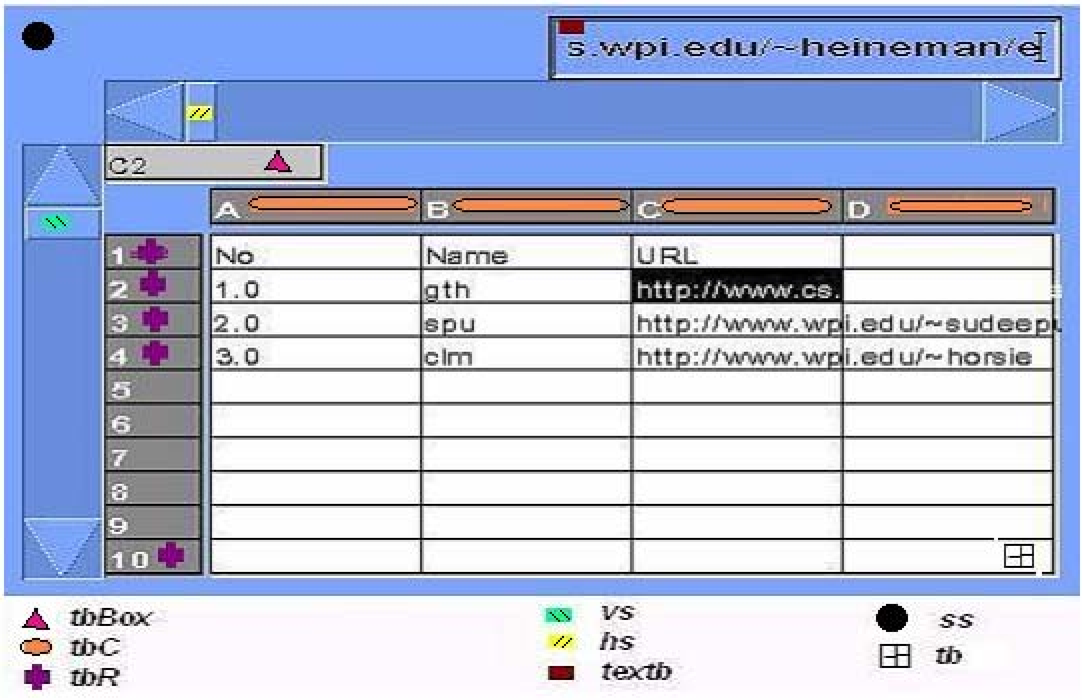
C h a p t e r   3

CASE STUDIES

3.1 SPREADSHEET APPLICATION

We use the following motivating example of the Java bean based architectured spreadsheet component as our case study. This spreadsheet application was available from previous research, ADAPT project. This application is a good candidate to be adapted and can be treated as an application composed of blackbox components. We have already put in hooks [3, 4, 5, 6] for adaptations, which were manually inserted in the form of supporting code and we can thus call this candidate component an adaptable component.

As shown in Figure 3.1, this spreadsheet application consists of eight interacting beans. The Tablebean *tb* displays the matrix of information in the form of C columns and R rows, which are represented by a small grid icon. There are two scrollbar beans for horizontal scroll *hs* and for vertical scroll *vs*, which are represented by small rectangular icons with lines on them. We have row header TableBeans for row *tbR* and for column *tbC, which* are represented by the oval and plus sign. There is a TableBean *tbBox* which displays the current position /selection status denoted by the triangle. Also there is a TextBean *textb*, which allows the user to input the values for the spreadsheet at current cell selection. A small rectangular box denotes it in Figure 3.1. Finally there is an invisible SpreadSheetBean *ss* denoted by a dot; it maintains and calculates all the values in the given spreadsheet.



	A	B	C	D
1	No	Name	URL	
2	1.0	gth	http://www.cs.	
3	2.0	spu	http://www.wpi.edu/~sudeepu	
4	3.0	clm	http://www.wpi.edu/~horsie	
5				
6				
7				
8				
9				
10				

Figure 3.1: Spreadsheet Application Bean like interactions.

All these beans are created from within an applet called FinalApp, which registers the interactions between the beans. The communication triggers on the basis of mouse events. For example when a user selects a cell in table bean tb, a TableEventObject event is generated by the tb. The FinalApp further processes this event by setting entry in the corresponding cell, the value is updated in the text bean box textb and the current cell entry is also update earlier in the tbBox by the FinalApp.

Thus this type of integrated collaboration implies that this is a typical example of component–based software system. It also satisfies our requirement of having a black box software system for our adaptation framework. We thus see this application as a whole Spreadsheet Application, which now requires being adapted for additional functionality.

3.2 ADAPTATIONS PROPOSED

When the Spreadsheet components shipped as a blackbox component to the application builder, one can have detailed knowledge about this Spreadsheet component in form of the CSL script. The CSL script describes the Spreadsheet component and also suggests which portions of the systems (methods, components) are adaptable.

Further as suggested one can change this CSL script or write a new CSL script to impose the required adaptations for this software system.

These kinds of adaptations in Table 3.1 are chosen to highlight different features of our framework.

No	Adaptation	Changes in application/Use
A1	Defining notification functions are to be invoked when a particular cell changes.	Compare if the old value of cell is changed or not.
A2	Altering spreadsheet to only send updated cells visible to bean showing current selection	Avoid those refresh events that take place for all the spreadsheet.
A3	Defining new functions for spreadsheet to use like e.g. if an URL is input it should have some special handling over a string and so on.	Return some portion of URL specified or the text of the URL can be a number for current stock price of a company.

Table 3.1: Adaptations chosen for framework.

These adaptations are implemented to test the validity of our approach towards adapting the blackbox components for additional features.

3.3 ADDITIONAL ADAPTATIONS PROPOSED

We try to recognize and implement new adaptations with respect to this spreadsheet application and try to implement them for evaluation purpose.

No	Adaptation	Brief Description /Use
A4	Auto entry (date time) for each entry, i.e. whenever spreadsheet content is changed, a track of change is kept.	Works like a log for keeping information about the cell changes and
A5	Multiple Spreadsheets, ability to shift to different sheet based on an event. also ability to do reference between the spreadsheets.	Like multiple sheets in excel or other commercial spreadsheets. Referencing here indicates one can refer to a particular spreadsheets value in other. e.g. sheet2:B2
A6	Scan Input of textbox for email address, if valid email address say add to the address book	Filter like adaptation to do some meaningful operation on the inputs to the spreadsheet.
A7	Adapt spreadsheet to display a picture in a given cell.	If input string is a jpeg, gif file with appropriate path/reference, display the picture in the TableBean tb

Table 3.2: Adaptations proposed for evaluation.

These additional features are specified to be able to gauge our framework and approach for the evaluation purposed discussed in detail in the Evaluation Chapter 5.

3.4 COMPONENT MODEL IMPLEMENTATION

Component Model Implementation (CMI) [1], is a component deployment strategy in which we deploy all the components as shown in figure 3.3

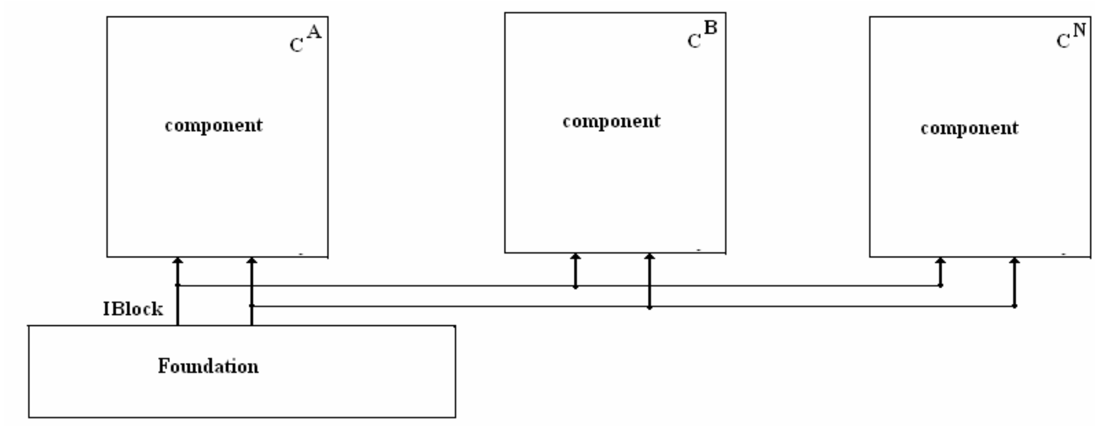


Figure 3.2: Component Model Implementation.

Here we have the components that follow an interface-based pattern, which enables them to communicate with each other. The components are instantiated and connected to each other at run time by the container “Foundation” and after all components are properly instantiated the application runs. Thus the Foundation controls the life cycle of this component-based application.

We use this model as our case study for deployment of the component and propose to integrate our adaptation technique with this approach. This approach is shown in Figure 3.3

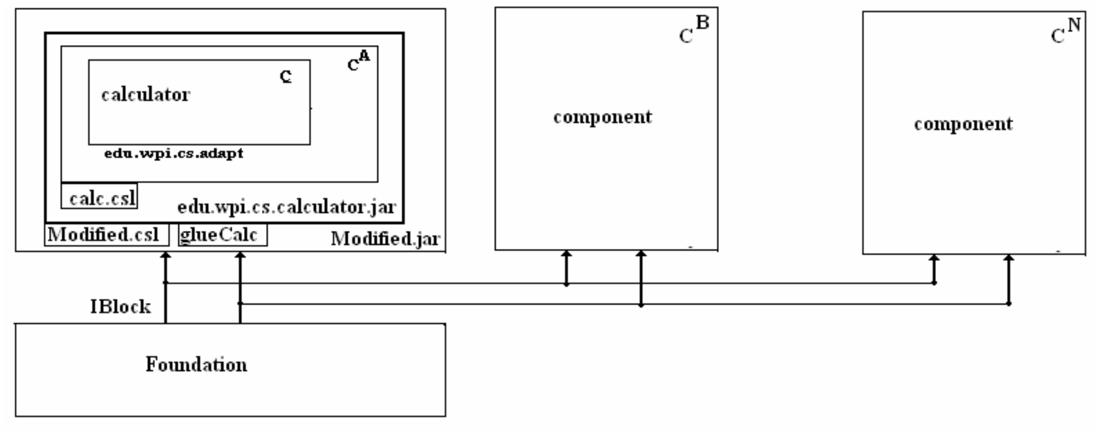


Figure 3.3: Component Model Implementation with adaptation technique.

Thus now using the approach suggested in this thesis the container of CMI parses the concerned CSL files, gets the concerned glue code and inserts the callbacks according to the information suggested in the corresponding CSL script. Thus we have a newly adapted component deployed in the CMI environment .A detailed explanation of this adaptation approach using CMI is given in Chapter 5.

3.5 ENTERPRISE JAVA BEANS

We would also like to discuss the work of integrating our adaptation techniques with the EJB [20] environment. EJB is one of the popular component models chosen by many application assemblers since it supports the distributed environment.

Our previous research by Kanetkar [8] focuses on the adaptations with respect to EJB. It also proposes a design to enable run time adaptation using the EJB Technology.

3.6 SUMMARY OF CHAPTER

In this chapter we discuss about the case studies we select to prove our approach. Section 3.1 gives information about our motivating case study, the Spreadsheet Application.

Section 3.2 and 3.3 explains about the adaptations that were done for proving our approach, as well as for evaluation purposes. These adaptations are carefully chosen to demonstrate the flexibility, applicability and functionality of our approach.

Section 3.4 discusses another case study that of a component deployment model CMI. We, in this thesis, present the way one can integrate our technique with the component deployment model and here in this chapter we briefly introduce the CMI deployment model. Section 3.5 further talks about EJB , a popular component deployment technique.

C h a p t e r   4

DESIGN AND METHODOLOGY

4.1 INTRODUCTION

Reusing a component avoids implementing same functionality from scratch and minimizes maintenance cost [3] in most cases. Components to be reused are mostly black box components. This gives us motivation for presenting an environment and approach for adapting a blackbox adaptable component for new additional features. This will enable it to fit in the proposed new system.

The basic essence of the mechanism we are suggesting is that we can adapt *adaptable blackbox* components. We can make a component adaptable by either having the component designer use one of the tools or having the component adapter do it as specified in the assumptions section in Chapter 1.A component designer/adapter with a code level access can also use the AIDE tool to make that component into an adaptable component. Along with AIDE, a tool like CSL-Profiler, which returns the CSL specification unknown given black box component using the output of AIDE, may be used.

The extra functionally desired is coded into a glue code. Thus all the new code goes into glue code and as discussed in Chapter 3 we modify the CSL script for the given *adaptable component* to be adapted.

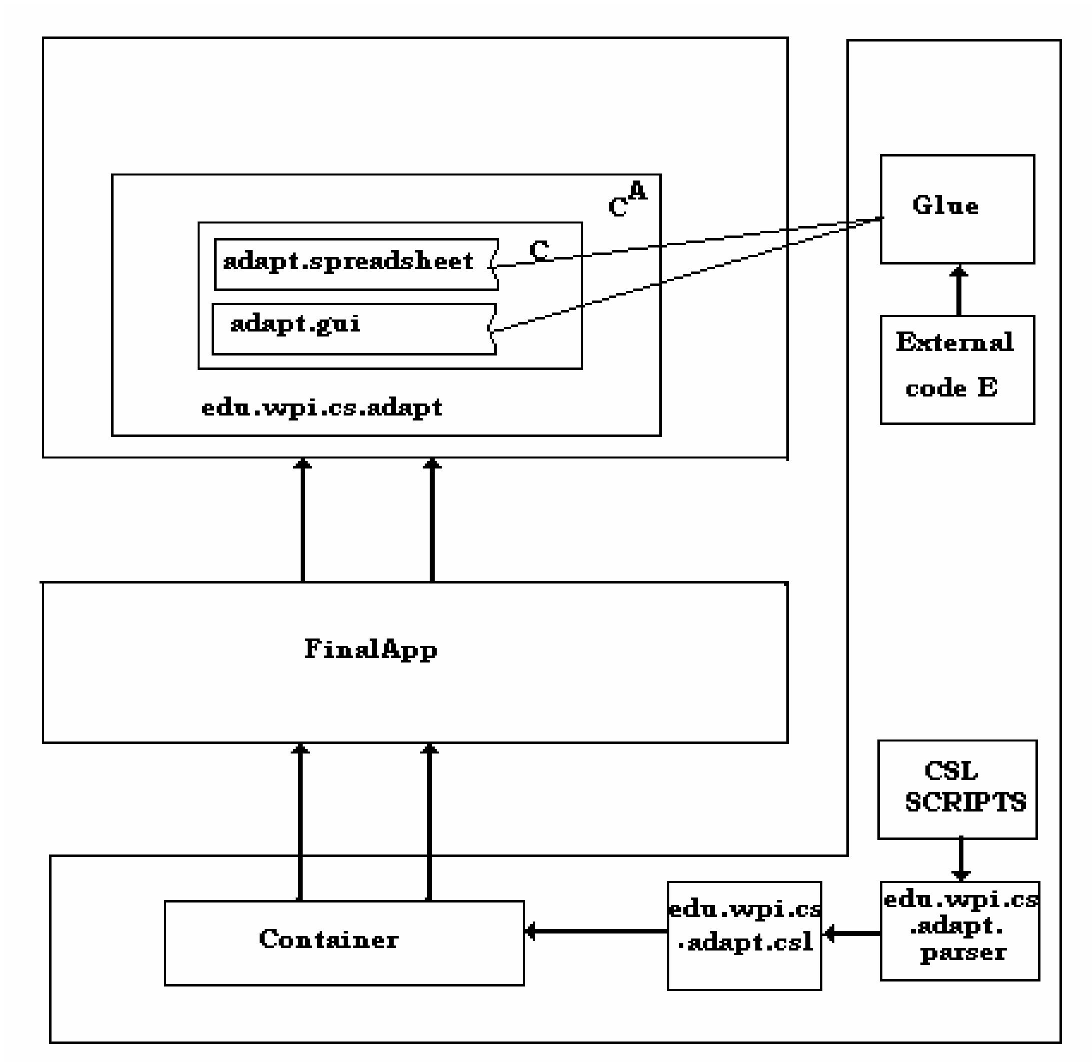


Figure 4.1: Component Deployment View: Spreadsheet.

As shown in Figure 4.1 we have a component C which using tools like AIDE can be converted to an adaptable component C^A. This component has its own glue code where the system assembler writes source code for the new adaptations. It also has it’s own CSL script file as discussed in Chapter 3. Container here invokes the parser to parse CSL script and the concerned information is updated in the parser structures. Using this information, the glue code is invoked and callbacks are set to monitor the events for the desired methods through the hooks present in the adaptable component.

Further in Figure 4.2 we can see the sequence, in which this process takes place,

1. Container fetches the CSL scripts
2. Container invokes CSL parser and parses the scripts
3. The parser generates the concerned CSL Object
4. Then object is returned to the container
5. Container invokes the FinalApp Spreadsheet application
6. From information present from CSL scripts container invokes the glue code
7. All necessary supporting classes are loaded and application runs with adaptations in effect at runtime.

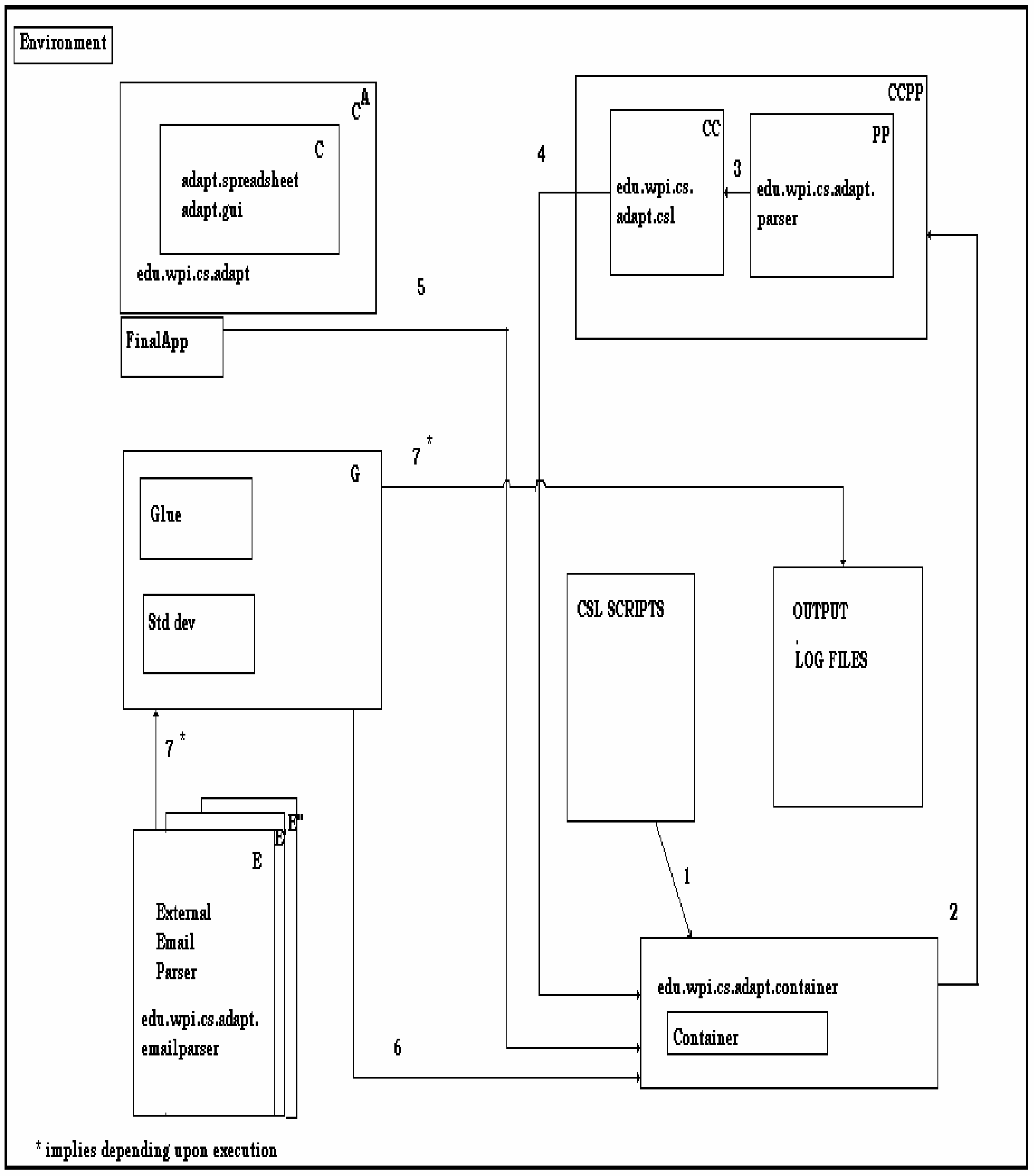


Figure 4.2: Interactions between Adaptations.

Figure 4.3 gives a brief idea of how things would work for multiple components to be deployed in similar fashion as that of the infrastructure presented in Figure 4.1.

The Container would work similar to the way discussed earlier in this section. The only difference now would be that instead of a single component multiple components would be loaded and there can be multiple number of glue files and CSL scripts local to the components.

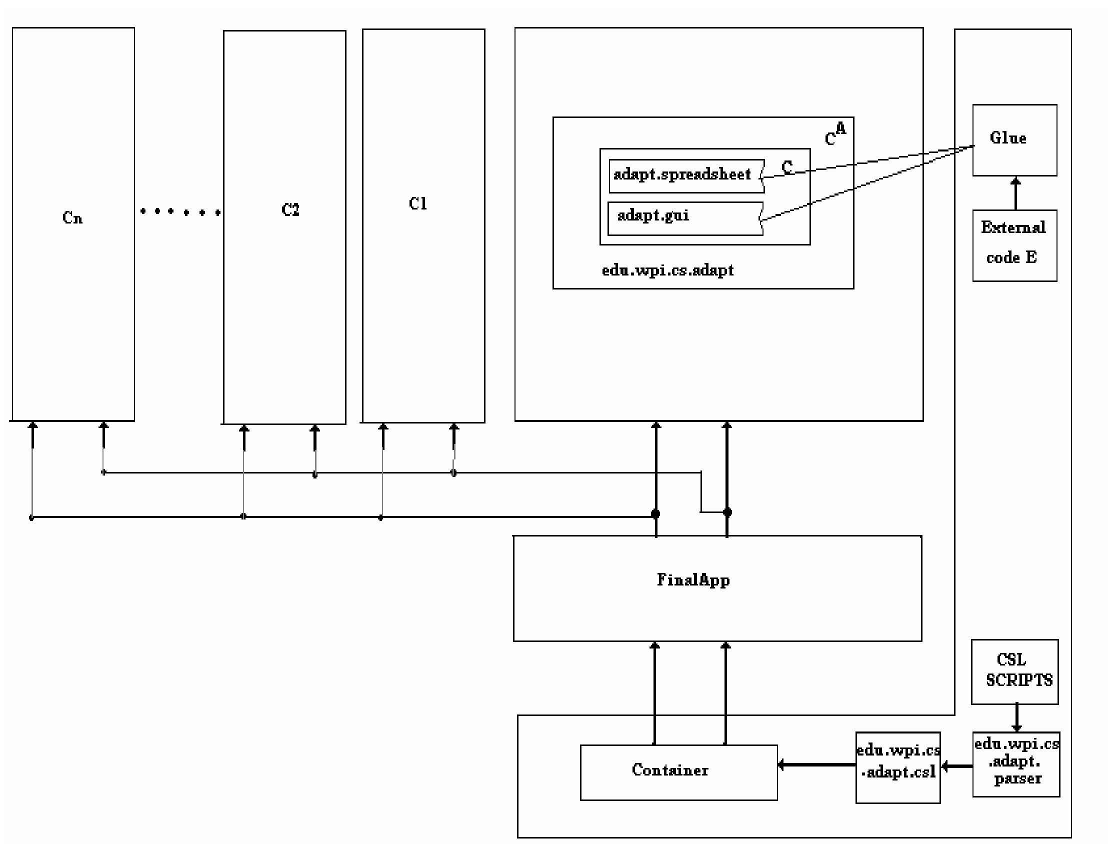


Figure 4.3: Component Deployment View: components like model.

4.2 TOOLS.

Our adaptation mechanism uses some tools that were developed as a part of our in-house research by fellow researchers. Application designers use these tools to instrument the non-adaptable component to adaptable component. Some tools help to generate the corresponding CSL of these adaptable component to help more in the adaptation process by having semantic CSL based information about a given black box adaptable component.

4.2.1 AIDE tool

We need adaptable components for our evaluations. We incorporate in our framework, AIDE, a technology to upgrade a software component to support Active Interfaces [10]. This technology helps us to identify the adaptations allowed by the given component using a customized tool. Thus the designers would now be able to select the methods in the component, they want to be adaptable and the concerned before-after callbacks would be installed.

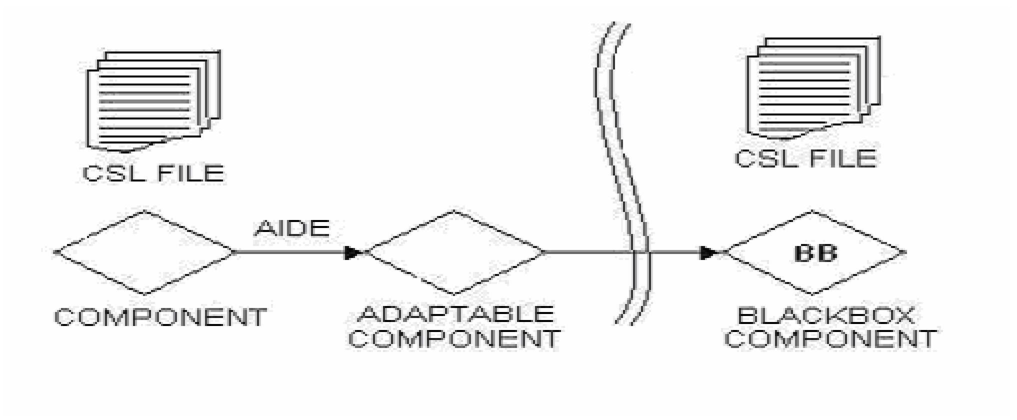


Figure 4.4: Profiler and AIDE tools.

AIDE has its own easy to use GUI that enables one to select the methods to be instrumented from a given application. Once the method is instrumented, some code is added appropriately to the application source, which makes that corresponding method adaptable. This newly added piece of code is just a hook inputted in the application source code to enable one to recognize the callbacks. As shown in Figure 4.4, an application designer can take a component and using AIDE package it to be an “Adaptable Component”.

4.2.1.1 USAGE: To use this tool follow steps given below

A. Run AIDE tool using command, java -jar ../extract-aide/aide-gui.jar

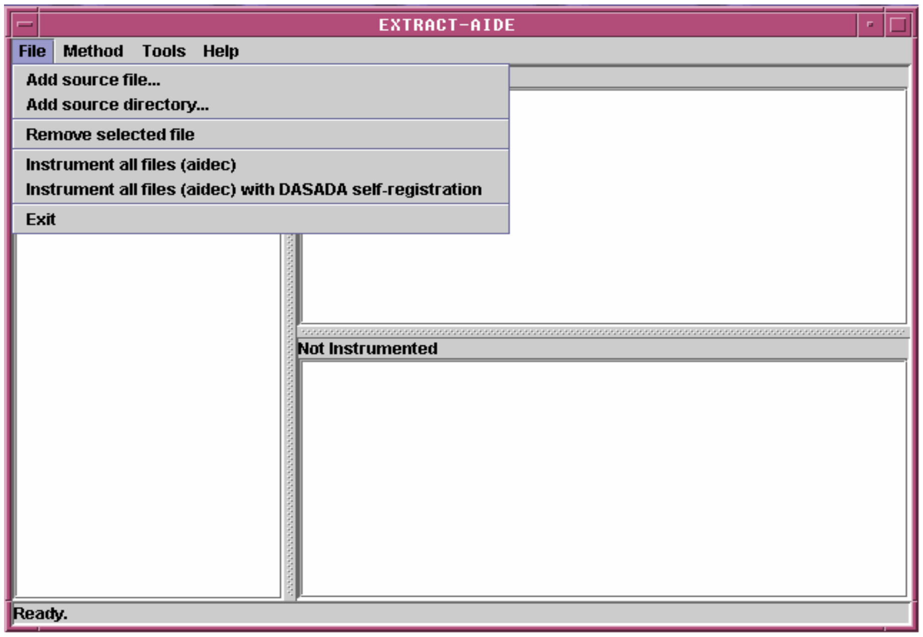


Figure 4.5: AIDE front end.

B Select the source code to instrument as shown in Figure 4.5 and 4.6

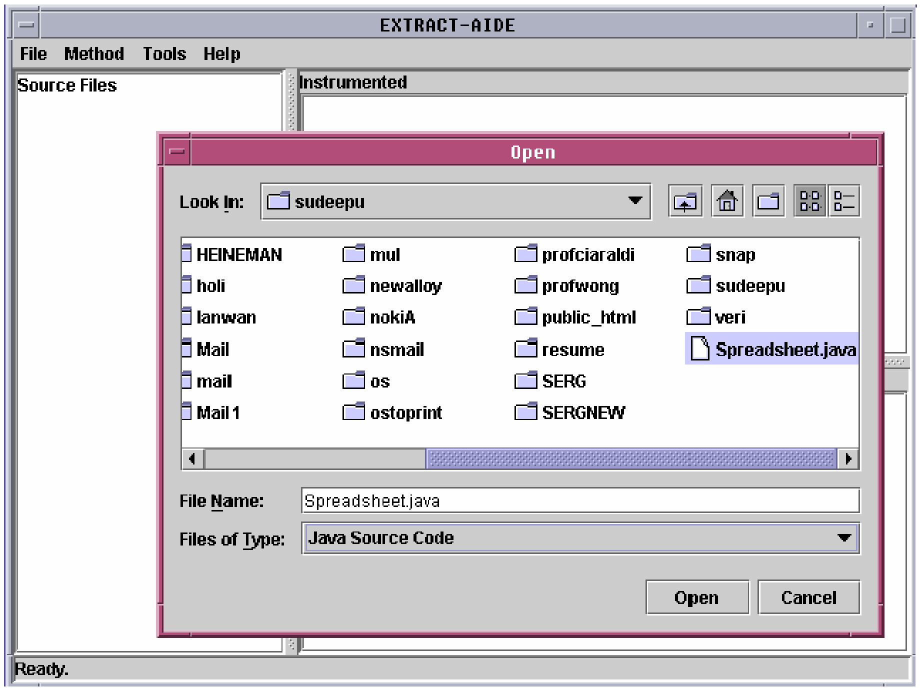
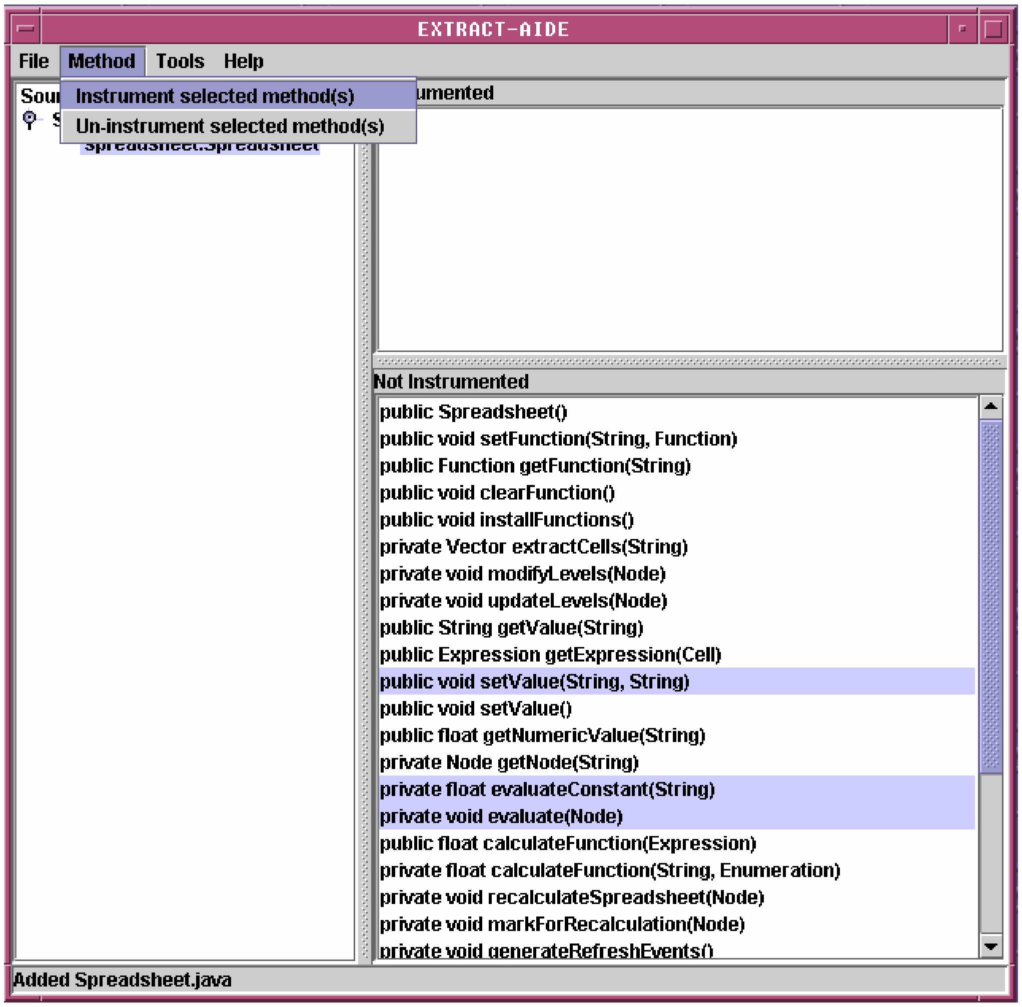


Figure 4.6: AIDE front end: select source.

C Select the methods you want to instrument (or un-instrument) and

D Select instrument option from Method menu as shown in Figure 4.7



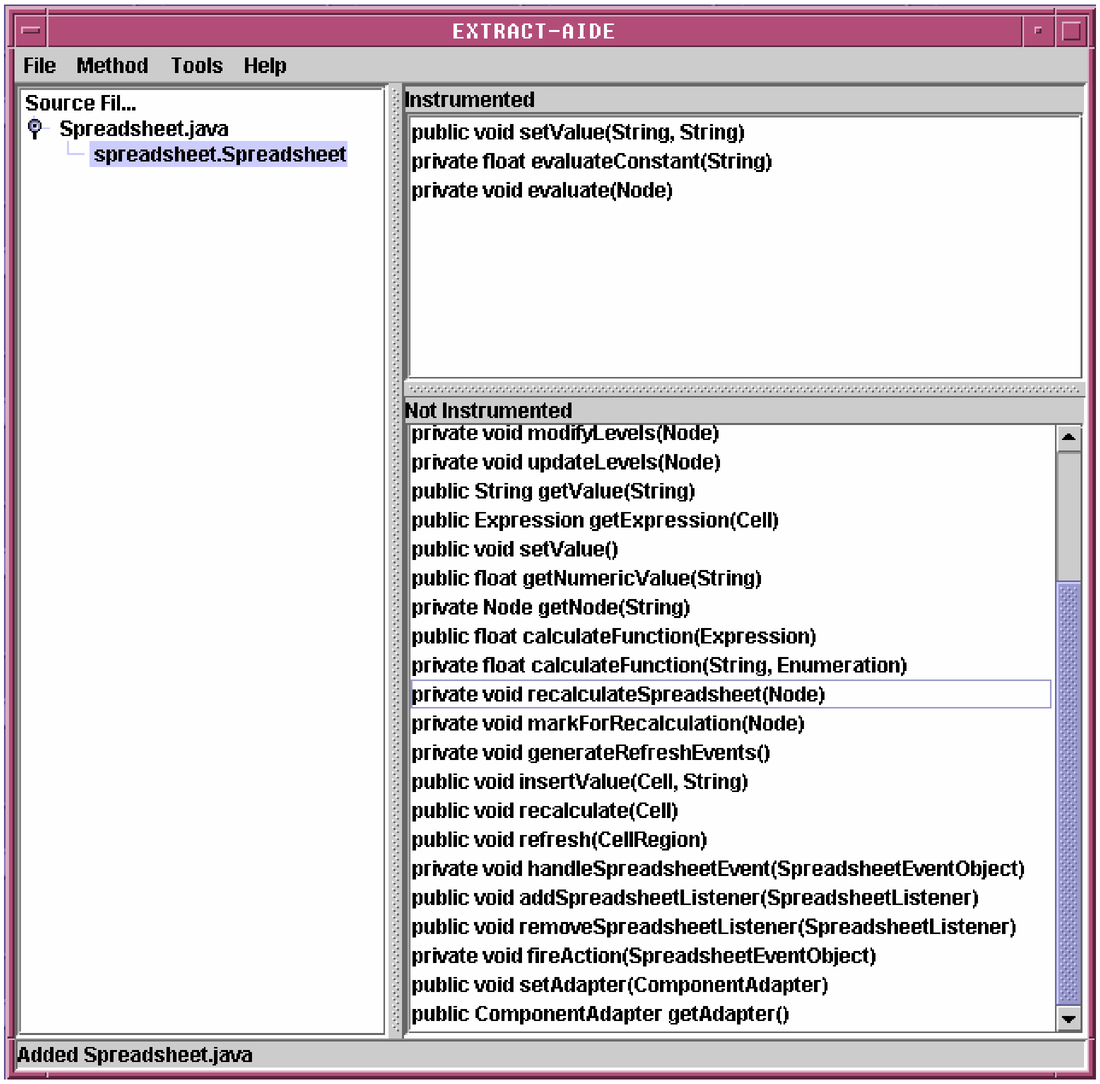
public Spreadsheet()
public void setFunction(String, Function)
public Function getFunction(String)
public void clearFunction()
public void installFunctions()
private Vector extractCells(String)
private void modifyLevels(Node)
private void updateLevels(Node)
public String getValue(String)
public Expression getExpression(Cell)
public void setValue(String, String)
public void setValue()
public float getNumericValue(String)
private Node getNode(String)
private float evaluateConstant(String)
private void evaluate(Node)
public float calculateFunction(Expression)
private float calculateFunction(String, Enumeration)
private void recalculateSpreadsheet(Node)
private void markForRecalculation(Node)
private void generateRefreshEvents()

Added Spreadsheet.java

Figure 4.7: AIDE front end: Instrument Method.

This will instrument the methods from the given source code and make necessary additions of hooks to the source code and will generate the instrumentation information xml file

E The instrumented method would be reflected in the tool GUI as shown below; also an instrumented method can be un-instrumented anytime.



public void setValue(String, String)
private float evaluateConstant(String)
private void evaluate(Node)

private void modifyLevels(Node)
private void updateLevels(Node)
public String getValue(String)
public Expression getExpression(Cell)
public void setValue()
public float getNumericValue(String)
private Node getNode(String)
public float calculateFunction(Expression)
private float calculateFunction(String, Enumeration)
private void recalculateSpreadsheet(Node)
private void markForRecalculation(Node)
private void generateRefreshEvents()
public void insertValue(Cell, String)
public void recalculate(Cell)
public void refresh(CellRegion)
private void handleSpreadsheetEvent(SpreadsheetEventObject)
public void addSpreadsheetListener(SpreadsheetListener)
public void removeSpreadsheetListener(SpreadsheetListener)
private void fireAction(SpreadsheetEventObject)
public void setAdapter(ComponentAdapter)
public ComponentAdapter getAdapter()

Added Spreadsheet.java

Figure 4.8: AIDE front end: After Instrumentation view.

4.2.2 CSL-profiler tool

The CSL-profiler [15] tool uses the information from the AIDE tool and generates the concerned CSL specification for that component or application describing the adaptations available to the application builders and/or concerned third parties.

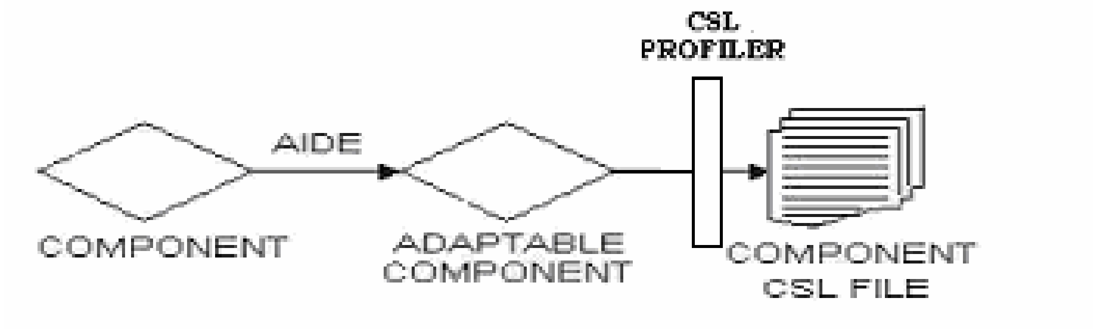


Figure 4.9: CSL Profiler tool.

As shown in Figure 4.9, the component designer creates an adaptable component-using AIDE and provides CSL specification of the adaptable properties. It is important to have the CSL specification for a given adaptable component, but it might not be always be helpful to create such CSL using a manual process. CSL-Profiler tool uses the AIDE output XML scripts (as shown in appendix A) and generate a CSL specification script from it that is understood by the environment we are proposing.

CSL-profiler along with generating CSL script for the component, also does some extra work by generating CSL keywords like ADAPT, FINAL appropriately. These keywords are very important as they suggest whether a given method or property is adaptable or not.

For example if some method is declared as *final*

```
final void generateCSL()
```

This suggests that the method generateCSL is not supposed to be adapted any further.

Thus in a way this tool helps an application builder to have an appropriate CSL description for the blackbox component that is to be adapted

4.2.2.1 USAGE

To use this tool follow steps given below

1. Run AIDE tool

   Get xml descriptions of the instrumented files as shown in appendix A.

2. Keep the *.xml files in a certain repository.

3. Call the convert application of the tool as given below:

```
>java convert  <directoryName> < fileName1 fileName2 ... fileNameN>
```

Example:

```
>java convert aide-xml INFO-adapt.spreadsheet.Spreadsheet.xml
                        INFO- adapt.spreadsheet.Spreadsheet1.xml
```

4. The equivalent CSL file will be generated in the specified directory

Steps 3 and 4 can also be done using GUI version of the CSL profiler.

The screen shots for the same are shown in Figure 4.10 to Figure 4.14.

The GUI is used as follows,

1. Open CSL Profiler GUI

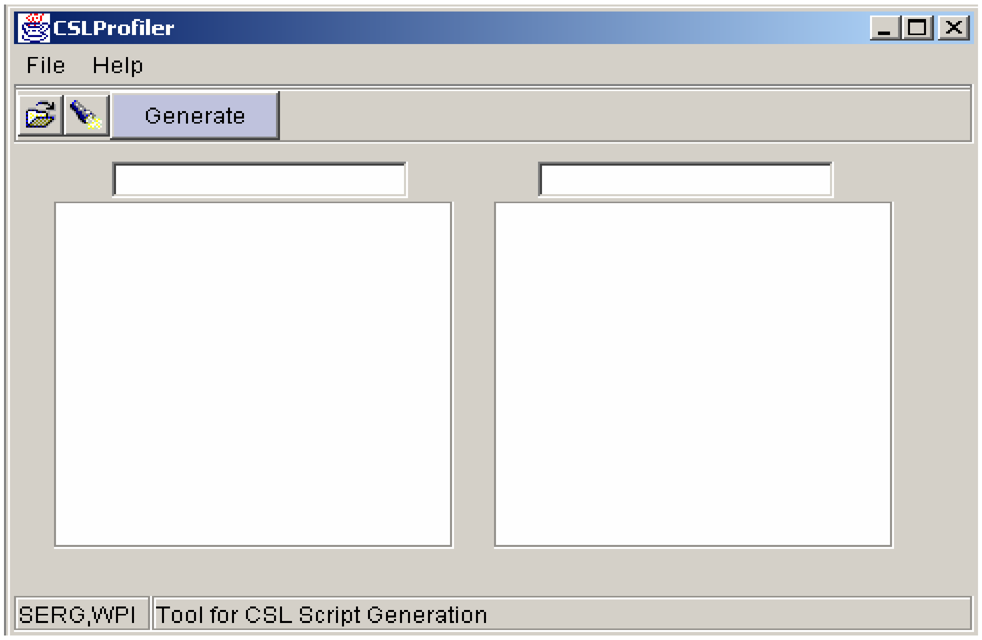


Figure 4.10: CSL Profiler tool GUI.

2.Select the Input xml file (generated by AIDE)

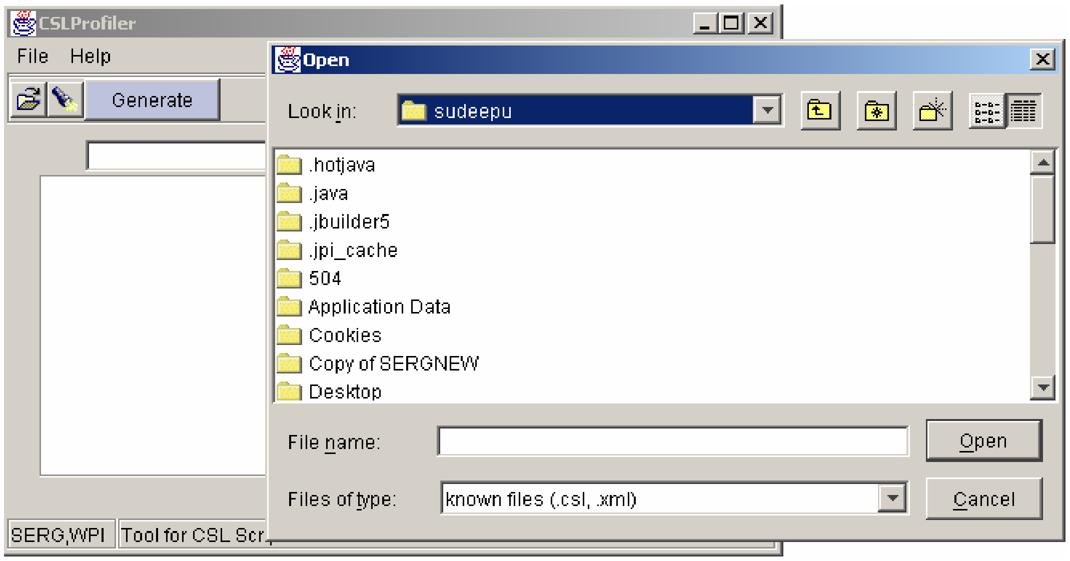


Figure 4.11: CSL Profiler tool Open File Menu.

3.Click Open to open selected file,

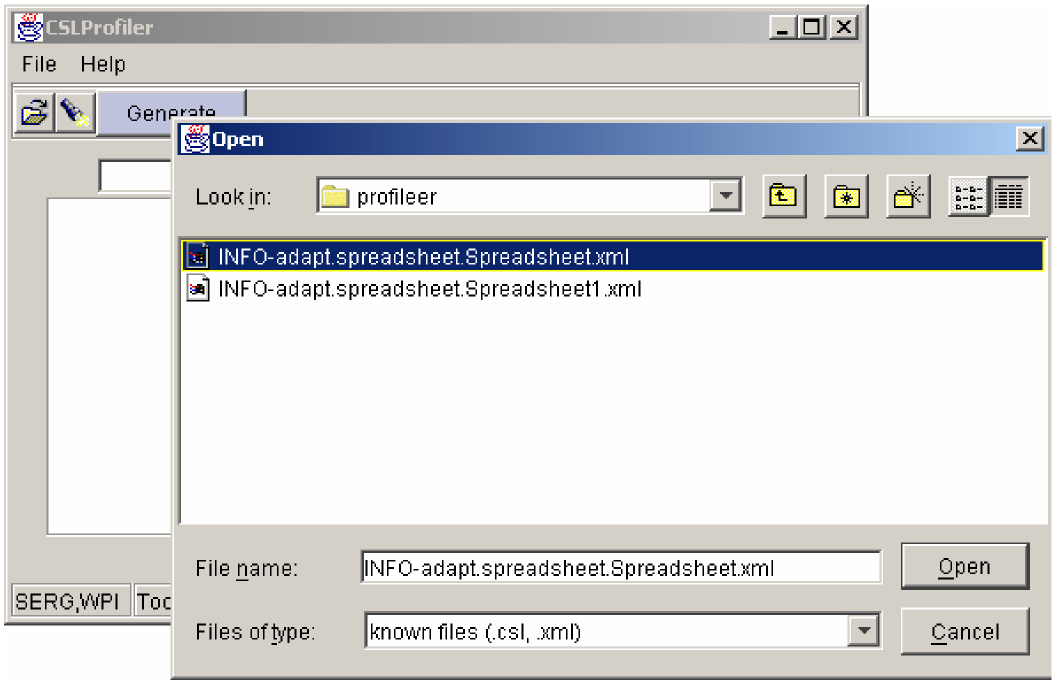
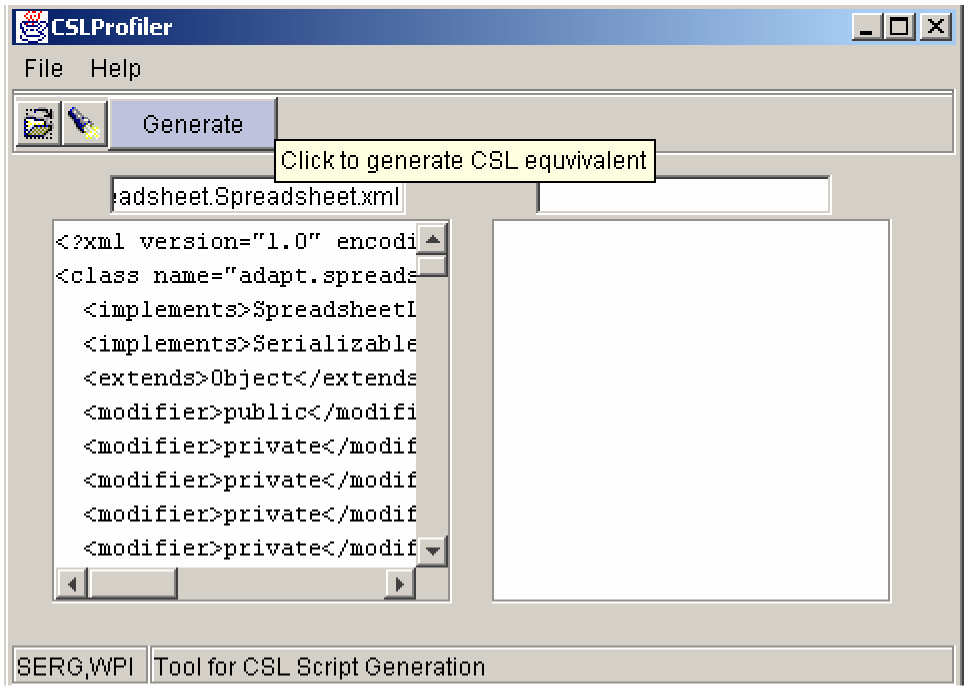

CSLProfiler
File Help
Generate
Open
Look in: profileer
INFO-adapt.spreadsheet.Spreadsheet.xml
INFO-adapt.spreadsheet.Spreadsheet1.xml
File name: INFO-adapt.spreadsheet.Spreadsheet.xml
Open
Files of type: known files (.csl, .xml)
Cancel
SERG,WPI
Too

Figure 4.12: CSL Profiler tool, opening Xml File.

4. Selected file will be reflected in the box on your left hand side, to generate equivalent CSL script click Generate button.


CSLProfiler
File
Help
Generate
Click to generate CSL equvivalent
SERG,WPI
Tool for CSL Script Generation

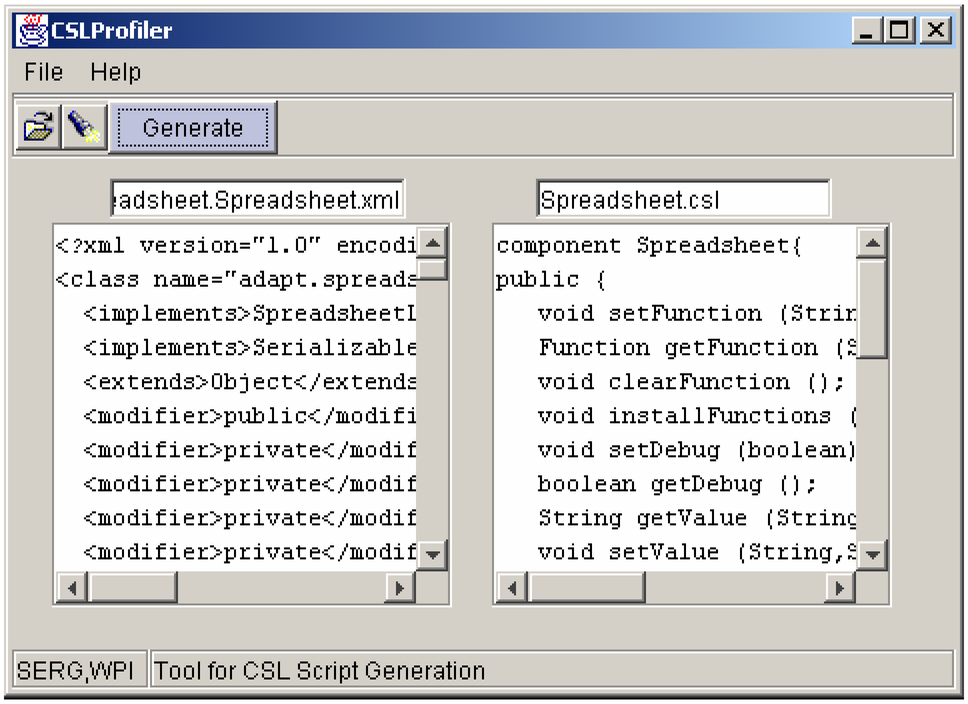
adsheet.Spreadsheet.xml

```
<?xml version="1.0" encodi
<class name="adapt.spreads
  <implements>SpreadsheetI
  <implements>Serializable
  <extends>Object</extends
  <modifier>public</modifi
  <modifier>private</modif
  <modifier>private</modif
  <modifier>private</modif
  <modifier>private</modif
```

Figure 4.13: CSL Profiler tool Generate CSL option.

<u>Note</u>: *Desired CSL script will only be generated provided that the xml file is an output of the AIDE tool or is similar to it. Appropriate error handling is done to ensure this restriction since here we are looking forward to recognize the instrumented members and other important members of the actual code from which the input xml document is generated (by AIDE).*

5.Generated Csl script will be reflected in the box on right hand side



```
<?xml version="1.0" encodi
<class name="adapt.spreads
  <implements>SpreadsheetI
  <implements>Serializable
  <extends>Object</extends
  <modifier>public</modifi
  <modifier>private</modif
  <modifier>private</modif
  <modifier>private</modif
  <modifier>private</modif
```

```
component Spreadsheet{
public {
   void setFunction (Strin
   Function getFunction (S
   void clearFunction ();
   void installFunctions (
   void setDebug (boolean)
   boolean getDebug ();
   String getValue (String
   void setValue (String,S
```

Figure 4.14: CSL Profiler tool Generated CSL script.

Generated Csl script will also be updated in a .csl file. The name of this file will be “classname.csl”. The class name here is the name of the class that was actually instrumented using AIDE and whose AIDE output is given as an input to the CSLProfiler.

The specification outputted by the CSL Profiler tool can be used for knowing more about the component especially while adapting it.

4.3 GLUE CODE:

Glue Code is an extra code written for the adapting the given component for a new or enhanced feature. Glue code typically consists a special lines of code where in logic with respect to the new /enhanced functionality for a given component to be adapted is actually implemented and which is later associated with this particular component like for example in CSL specification of the component we give the reference of the additional functional logic using the CSL *class* clause. Example

```
component adaptme adapts adapt.spreadsheet.Spreadsheet{

    class Glue;
    …
 }
```

This kind of Glue code is written generally for the black box type adaptation technique, the one we are suggesting.

For example, In our case study, if we need an adaptation such that, given there is a string input to the Spreadsheet application, we want to know if it is an URL and accordingly display the URL's contents.

We can write the corresponding functionality in our glue code in form of a method as ,

```
/**
 *Returns URl value as  output if the string is valid
 *URL or the same string itself
 */
public String beforeEvaluateC(edu.wpi.cs.adapt.Context c) {
       // String is the first argument
        Object args[] = c.getArgs();
        String s = (String) args[0];
     try {
            URL valueURL = new URL(s);//construct URL
            InputStream is = valueURL.openStream();
                                           //read URl
            BufferedReader br = new BufferedReader(new
                                     InputStreamReader(is));
            String value = br.readLine();//get URL value
            args[0] = value;
            return value;//return value changed
      } catch (MalformedURLException mfe) {
            return s;// not URl return value unchanged
      } catch (IOException ioe) {
                        return s;
            //cannot read URl return value unchanged
      } catch (Exception e) {
            return s;
           // exception return value unchanged
        }
}// end of beforeEvaluateC(Context)
```

Note: *We use a special input type Context to specify the input for a given glue method. This is done to ensure the chaining of the components in the system so as to ensure multiple adaptation of a same method .In our case one can notice this when we adapt the evaluateconstant() method of Spreadsheet for email adaptation , URL adaptation and the Log file adaptation .*

We can further include the information about this new functionality / method in the CSL Script of this particular component description as

```
component adaptme adapts spreadsheet{
        class Glue;
        action beforeEvaluateC (in java.lang.String);
        method float evaluateConstant (java.lang.String s){
             before beforeEvaluateC (s);
        };
     }
```

Here we suggest invoking method *beforeEvaluateC* from Glue code before invoking the actual method *evaluateConstant* from the original application.

Also one can see that method *beforeEvaluateC* is defined above as an action that takes string as an input.

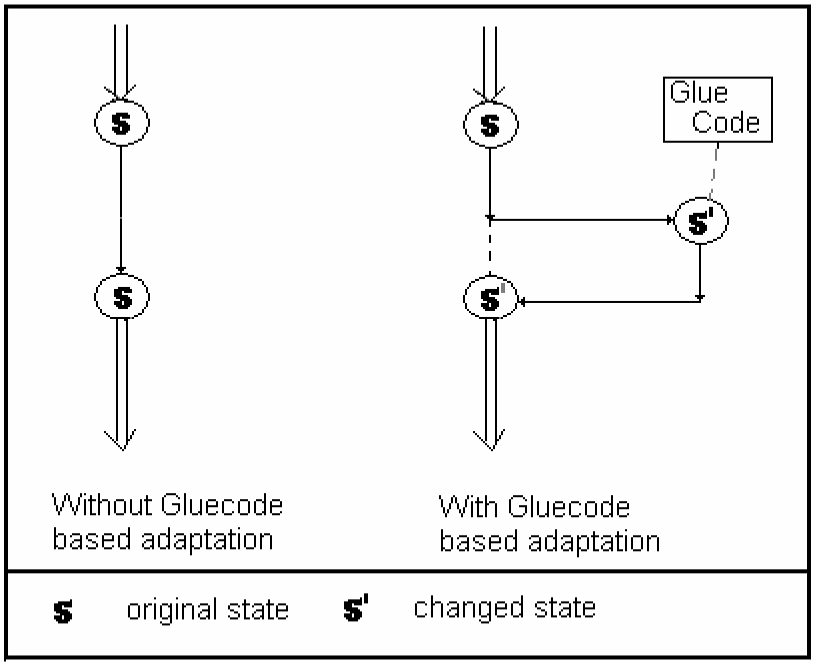

Glue Code
S
S'
Without Gluecode based adaptation
With Gluecode based adaptation
S original state
S' changed state

Figure 4.15: Application State based on Glue code.

In Figure 4.15, we consider the input String s that is an input to the original method *evaluateConstant*. When there is no adaptation as in first case of above diagram we can see that the state of the String s is not changed since we don’t do anything special here irrespective of whether the string is an URL or not.

In second case, we apply our technique of adaptation for the URL example stated above and use the glue code, where concerned functionality for detecting URL string input is coded. Here we can see that after adaptation the state of String s is changed to s’ accordingly w.r.t. the adaptation specified.

Note: *We assume that the input string is an URL else the status of input String s will remain unchanged*.

4.3.1 ADVANTAGES OF GLUE CODE

The basic advantage here is that for writing the additional functionality required for adapting a given black box component, the component internals are not required to be modified, instead one can write the code in form of glue code and carry out the adaptation.

Also advantage is that the adaptations are independent of each other (other adaptations) .One glue code can contain multiple adaptation functionality with respect to the component required to be adapted.

4.3.2 DISADVANTAGES OF GLUE CODE

Major disadvantage is that the Glue code should be written with precision and care should be taken while dealing with objects whose properties are not known. For instance, in above URL example we know that in original component, the input to the method evaluateConstant() is an String but had the input been an Object it would have been difficult to write a Glue code that is compatible with the original component simply because We cannot anticipate what type of object is expected during runtime as an input and if it is type-casted in the original component since here we have an black box component. There has to be some decision making work involved before writing glue code about whether or not the adaptation is possible for the required feature.

Still over all we feel this Glue code technique works well for the suggested framework simply because it enables us to a blackbox component adaptation.

4.4 BASIC ADAPTATIONS

4.4.1 CSL: Adaptation A1

No	Adaptation	Changes in application/Use
A1	Defining notification functions are to be invoked when a particular cell changes.	Compare if the old value of cell is changed or not.

Table 4.1: Adaptation For notification functions.

4.4.1.1 Description

We try and compare to see if the value of the current cell is changed or not. We have to specify two things here for this adaptation

- Before the evaluate(), record the old value
- After the evaluate(), compare the new value with the old value

So while writing glue code we try to write the csl script for the adaptation such that we the above two things are reflected and easily understood by the container, which is going to read this CSL script.

4.4.1.2 Adaptation procedure

We have to instrument the method evaluate() from spreadsheet

In glue code we have, beforeEvaluate()

```
/**
 * Used to store the value for a given node to check in
 * future if the old value is same as new value
 */
public int beforeEvaluate(edu.wpi.cs.adapt.Context c) {
        // Node is the first argument
        Object args[] = c.getArgs();
        Node node = (Node) args[0];
        // store value in hashtable
        float f = node.getNumericValue();
        values.put(node.toString(), new Float(f));
        return 0;
}//end of BeforeEvaluate(Context)
```

After Evaluate()

```
/**
 * Used to check for a given node if the old value is
 * same as new value
 */
 public void afterEvaluate(edu.wpi.cs.adapt.Context c) {
        // Node is the first argument
        Object args[] = c.getArgs();
        Node node = (Node) args[0];
        // compare new value against stored value
        Float newValue = new Float(node.getNumericValue());
        Float oldValue = (Float) values.get(node.toString());
        //remove the old value from the hashtable
        values.remove(node.toString());
        if (oldValue.equals(newValue))
            return;
        //send notify
        System.out.println("Node: " + node.toString() + " changed
value");
    }//end of afterEvaluate(Context)
```

We have the required functionality coded in Glue.java, with beforeEvaluate( ) and afterEvaluate( ) being the concerned methods in the Glue code, we have necessary csl as,

```
component adaptme adapts spreadsheet{
      Class Glue;
      action beforeEvaluate (in adapt.spreadsheet.Node);
      action afterEvaluate (in adapt.spreadsheet.Node);
      method void evaluate (adapt.spreadsheet.Node node){
                  before beforeEvaluate (node);
                  after  afterEvaluate (node);
      };
  };
```

4.4.2 CSL: Adaptation A2

No	Adaptation	Changes in application/Use
A2	Altering spreadsheet to only send updated cells visible to bean showing current selection	Avoid those refresh events that take place for all the spreadsheet.

Table 4.2: Adaptation for generating refresh events.

4.4.2.1Description

Alter spreadsheet (*ss*) to only send to table bean (*tb*) updated cells visible to *tb*. Filter update messages to spreadsheet, so only the current visible cells of GUI are refreshed and relevant update messages generated

Here we specify a negotiate policy for this adaptation called RefreshPolicy. So while writing glue code we try to write the csl script for the adaptation such that we the above adaptations are reflected and easily understood by the container, which is going to read this CSL script.

4.4.2.2 Adaptation procedure

We have to instrument the method generateRefreshEvents() from spreadsheet .

In glue code we have, filterCells()

```
/**
 *  Used for refreshing the cells visible to the screen,
 */
public void filterCells(edu.wpi.cs.adapt.Context c) {
        // Vector is the first argument
        Object args[] = c.getArgs();
        Vector v = (Vector) args[0];//vector of cells to refresh
        int col, row;// row and column
        CellRegion visibleCells = fa.getVisibleCells();
         //get cells
        for (Iterator it = v.iterator(); it.hasNext();) {
            Node n = (Node) it.next();//get node

            col = n.getCell().getColumn();//get column of node
            row = n.getCell().getRow();   // get row of node
            if ((col < visibleCells.getStart().getColumn())
                || (col > visibleCells.getEnd().getColumn())
                || (row < visibleCells.getStart().getRow())
                || (row > visibleCells.getEnd().getRow())) {
                it.remove();
                //remove if any cell out of visible domain
            }//end of if
        }//end of for(..)
        return;
 }//end of filterCells(Context)
```

We have the required functionality coded in Glue. java, where in there is a concerned method filterCells( ) , we have necessary csl as,

```
component adaptme adapts spreadsheet{
      class Glue;
      action filterCells(refreshList java.lang.Vector);
      method void generateRefreshEvents(java.lang.Vector
                                           refreshList){
                  negotiate RefreshPolicy:
                  filterCells(refreshList);
      };
};
```

4.4.3 CSL: Adaptation A3

No	Adaptation	Changes in application/Use
A3	Defining new functions for spreadsheet to use like e.g. if an URL is input it should have some special handling over a string and so on.	Return some portion of URL specified or the text of the URL can be a number for current stock price of a company.

Table 4.2: Adaptation A3 for scanning input for URL.

4.4.3.1 description

If a cell contains a URL we can evaluate the string before evaluateConstant() and return the value of URL instead.

So while writing glue code we try to write the csl script for the adaptation such that we the above adaptation is reflected and easily understood by the container, which is going to read this CSL script.

4.4.3.2 Adaptation procedure

We have to instrument the method evaluateConstant() from spreadsheet.

In glue code we have, beforeEvaluateC()

```
/**
 * Returns URl value as output if the string is valid URL
 * or the same string itself
 */
 public String beforeEvaluateC(edu.wpi.cs.adapt.Context c) {
       // String is the first argument
        Object args[] = c.getArgs();
        String s = (String) args[0];
        try {
            URL valueURL = new URL(s);//construct URL
            InputStream is = valueURL.openStream();//read URl
            BufferedReader br = new BufferedReader(new
                                  InputStreamReader(is));
            String value = br.readLine();//get URL value
            args[0] = value;
            return value;//return value changed
        } catch (MalformedURLException mfe) {
            return s;// not URl return value unchanged
        } catch (IOException ioe) {
            return s;//cannot read URl return value unchanged
        } catch (Exception e) {
            return s;// exception return value unchanged
        }
}// end of beforeEvaluateC(Context)
```

We have the required functionality coded in Glue.java, with

beforeEvaluateC () being the concerned method in the Glue code,

We have necessary csl as,

```
component adaptme adapts spreadsheet{
        class Glue;
        action beforeEvaluateC (in java.lang.String);
        method float evaluateConstant (java.lang.Strings){
                    before beforeEvaluateC (s);
        };
};
```

4.4.3.2 Adaptation Output

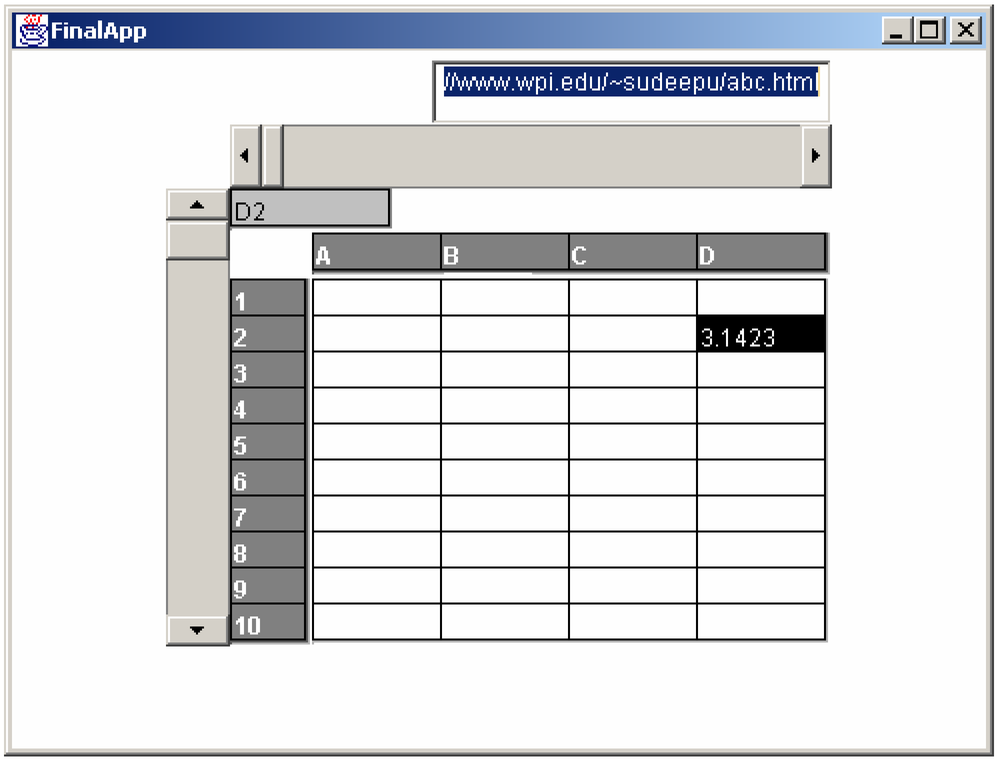


Figure 4.16: Spreadsheet URL Adaptation.

Figure 4.16 shows the output for the URL adaptation. We input a valid URL, which returns the value of Pi. When this URL is typed as a string input to the spreadsheet, its validity is checked and the content is assigned to the cell.

Note: *Here the URL selected for demonstration returns a single value. This was done to ensure that there is not much garbling of characters in the spreadsheet.*

4.5 CONTAINER:

We are seeking a Black box component-based software system as a candidate for adaptation. Thus we are not modifying any part of the code from the original component as it would certainly make this technique fall into white box category and the whole essence of this approach will be lost. This leads us to the new requirement of having a wrapper like component on t he top of now adapted component that understands the original component working and the new adaptations done using the Glue code. It implies that this component should understand Component Specification Language (CSL) and also deployment strategy used for example java beans [8] like or CMI (Component Model Infrastructure)[12] environment.

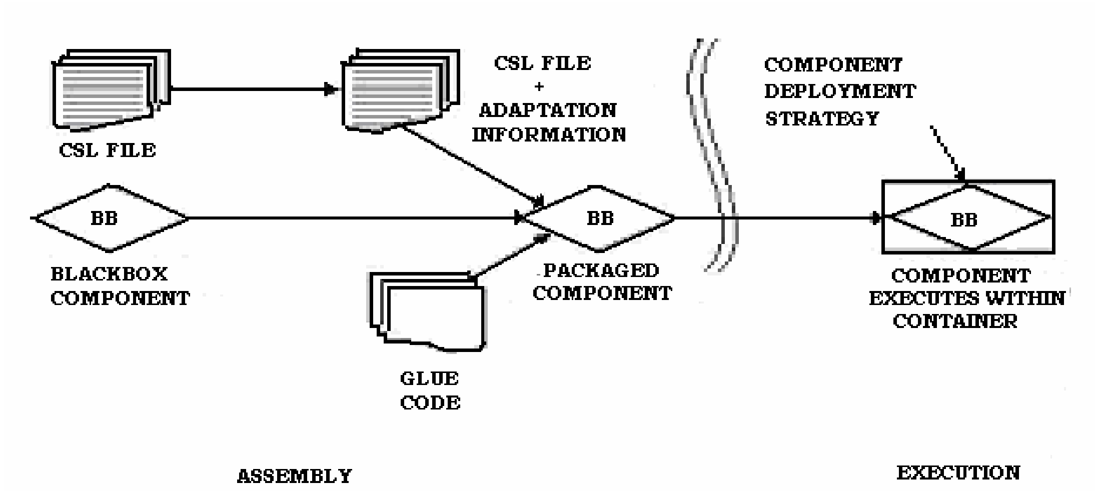

CSL FILE
CSL FILE + ADAPTATION INFORMATION
COMPONENT DEPLOYMENT STRATEGY
BB
BLACKBOX COMPONENT
BB
PACKAGED COMPONENT
BB
COMPONENT EXECUTES WITHIN CONTAINER
GLUE CODE
ASSEMBLY
EXECUTION

Figure 4.17: Whole Process of Component Adaptation.

We have to be very careful while understanding the needs of this component that we call as a “Container”. For example, in our case study we have a file called Final App, which is the principal applet for the Spreadsheet application. So if we have to adapt this Spreadsheet application, we write appropriate glue code and CSL scripts. CSL scripts describe this application and suggest / impose the adaptations to be done.

We write a container component for this adaptation say “container applet”. This Container applet typically consists of mechanism that takes CSL script file as input and invokes the corresponding parser to get the information regarding the CSL in form of some data structure. Using the data structure then this container component tries to invoke the application main file/applet by instantiating it using technology like Java Reflection and then again check for the portions of the application that are to be adapted using the CSL script and insert concerned callback as per suggested in CSL script. The Glue code and the application now works together to form a new blackbox component with additional desired features that were added by adapting the original application. Instead of the original main execution file like FinalApp, now the container component becomes the main execution file of this adapted application.

For example

With respect to our Spreadsheet example, in container applet

First we take an input csl file and parse it

```
 // invoke the parser
    CSL.main (cslFileinput);
```

Then we get the Object of the Data structure where the information from CSL-parsing is now Stored

```
// Get the application
  CSLApplication app = CSL.getApplication();
```

Container Applet, now using Reflection, We get the object of FinalApp, using the information from the CSL. We check for the Components to be adapted by going through the Structure where we now have the information from the CSL file.

Example of an Adaptation for a given Component

In Container Applet, Consider the example we used

```
component adaptme adapts spreadsheet{
   class Glue;
   action beforeEvaluateC (in java.lang.String);
   method float evaluateConstant (java.lang.Strings){
          before beforeEvaluateC (s);
   };
}
```

Using reflection we get the Glue class and traversing through data structure. We get the method *evaluateConstants* adaptation properties I.e. before this method is invoked, we have to set a callback that another method *beforeEvatuateC()* takes the charge and so the necessary adaptation for the given specification.

In Container Applet, We get the concerned other information about the before property and set a callback as

```
tca.insertCallback (tca.BEFORE,method_name,adaptObject,
       glueClass.getDeclaredMethod (bfmname, paramTypes));
```

This Container Applet understands how these components are deployed .If we are using a specific strategy it need to know about it.

4.6 DEPLOYMENT STRATEGIES

For presenting any infrastructure, it is important to stress on the deployment strategy to be followed. This is thus one of major factor when it comes to make a decision whether for the concerned problem the given framework is appropriate or not.

We present this framework for adaptation of the adaptable components, which are focused to be the part of the new system to be assembled. Thus the deployment strategy that we want to enforce for doing so should be in accordance with the actual deployment strategy of the new system. To present our view, we choose for example a component deployment strategy, CMI [1].

CMI is deployed in a manner that all the components of the new systems are places in a particular repository .The actual container called “Foundation” is in a different repository, scripts are used to specify the location of the blocks (components, e.g. jar files of the components) and those scripts are too placed in different repository.

Thus we can deploy   CMI as given in figure 4.18 for a book selling system called “book buyer” (actual example from CS509 class fall 2001, CS, WPI).

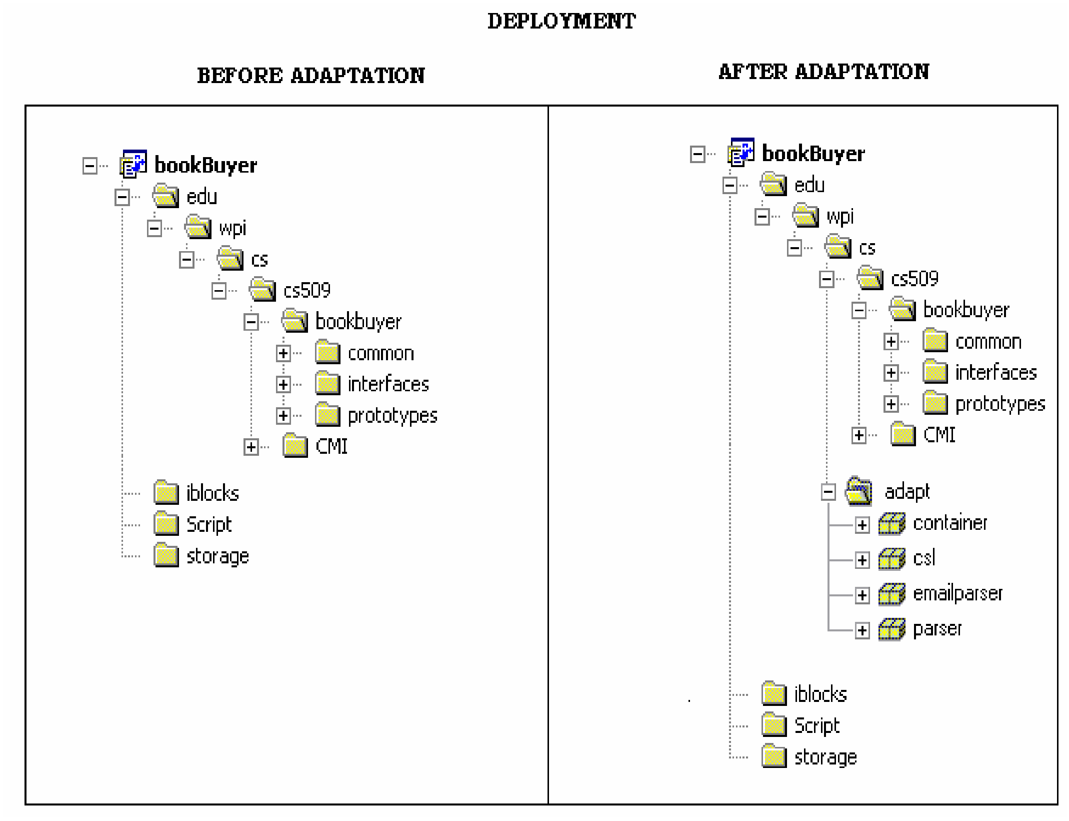


Figure 4.18: Deployment in CMI.

In figure 4.18 left hand side indicates the actual deployment of a system in CMI without any adaptation. We propose using our technique the deployment would be similar to that as shown on right hand side of Figure 4.18.

4.6.1 Elements required for adaptive deployment of environments like CMI:

We are proposing following elements needed for the adaptation for deployment of our framework with available techniques like CMI,

- Different container for the adaptation or editing Foundation for recognizing the provided CSL mechanism, setting Adapter and adding callbacks.
- Adding the “edu.wpi.cs.adapt.csl” package to ensure access to data structures provided.
- Adding the “edu.wpi.cs.adapt.parser” package provided to ensure valid parsing of the CSL script and population of corresponding structure.
- Adding the “edu.wpi.cs.adapt” package provided to enable having Adaptable components using active interface based technology.

Note: *Here we are not changing any deployment of the original approach only adding to it, this ensures that the application assembler can always resort to the original deployment plan ignoring the extra adaptation repository and simply enable or disable adaptations by using appropriate CSL scripts. Thus rightfully controlling the run-time behavior of this component based system.*

4.7 SUMMARY OF CHAPTER

In this Chapter we present the design methodology for the adaptation procedure we are suggesting. We briefly introduce the procedure in Section 4.1.Futher we introduce to the deployment issues of our case study, the Spreadsheet application. Also we provide a detailed diagram that's gives detailed steps about the adaptation procedure.

Section 4.2 talks about the tools AIDE and CSLProfiler. Section 4.2.1 talks about AIDE a tool that assists in converting a component to an adaptable by instrumenting its methods. Section 4.2.2 talks about the CSLProfiler tool that takes the output of the AIDE to generate corresponding CSL specification of the adaptable component. Detailed usage of both tools is provided in the respective section.

Section 4.3 talks about the Glue code and its use in detail. It also discusses about the advantages and disadvantages of the Glue code approach. Section 4.4 gives in detail how the basic adaptations are carried out. Section 4.5 describes how container works and how callbacks are inserted on the fly.

Section 4.6 describes in deployment strategy for integrating our approach with CMI component deployment strategy.

C h a p t e r   5

EVALUATION

5.1 INTRODUCTION

Evaluation of a system or infrastructure is one important measure of the practicality of that system/approach. Evaluation of the proposed infrastructure was ideally to be done keeping other adaptation techniques in mind. One form of evaluation would have been to compare different approaches depending upon various metrics. This scenario is difficult because the adaptations have different assumptions and are made in controlled environment as yet and follow no standard. To say in other words the adaptations are very application and end user specific.

We decided to evaluate the framework on the basis of several adaptations. These adaptations were carefully chosen to demonstrate the flexibility of the framework and test if the infrastructure is applicable in practice.

To evaluate our infrastructure, we required some metrics. We planned to evaluate design aspects too (qualitative). We proposed to evaluate software adaptations using following metrics, which we would consider while evaluating our approach as given in Table 5.1

Metric	Expectation with respect to Adaptation
Transparency	A component adaptation should be transparent i.e. a user should be unable to derive any difference in using the original component and adapted component.
Blackbox	A component should be blackbox for adaptation to avoid change of internals of the component itself like most of COTS.
Composablity	Adaptation technique should be composable with the component it is applied on, it should also be composable to other components as it was before adaptation and it should be composable with other adaptations.
Performance	Performance of the adapted component should be more or less same as that of the equivalent rewritten component.
Deployment	Infrastructure should provide how to deploy components; measuring overlays, tool support so on, can do this.

Table 5.1: Metrics proposed for evaluation.

The criteria in Table 5.1 are a subset of the relevant criteria of interest during Adaptation [2]. We need to develop scenarios by which these criteria will be evaluated. We developed several candidate adaptations to make on the spreadsheet component to compare the effectiveness of adaptation against the desired properties listed above.

5.2 CANDIDATE ADAPTATIONS FOR EVALUATION

No	Adaptation	Brief Description /Use
A4	Auto entry (date time) for each entry, i.e. whenever spreadsheet content is changed, a track of change is kept.	Works like a log for keeping information about the cell changes.
A5	Multiple Spreadsheets, ability to shift to different sheet based on an event. also ability to do reference between the spreadsheets.	Like multiple sheets in excel or other commercial spreadsheets. Referencing here indicates one can refer to a particular spreadsheets value in other. e.g. sheet2:B2
A6	Scan Input of textbox for email address, if valid email address say add to the address book	Filter like adaptation to do some meaningful operation on the inputs to the spreadsheet.
A7	Adapt spreadsheet to display a picture in a given cell.	If input string is a jpeg, gif file with appropriate path/reference, display the picture in the TableBean tb

Table 5.2: Adaptations proposed for evaluation.

The proposed adaptations in Table 5.2 were worked out in a similar way we would expect an application assembler to work them out. Doing so helped us to evaluate the practical problems faced in doing these adaptations and also helped to analyze how fruitful this approach can be for black box adaptations.

Additional Spreadsheet Adaptation Cases & Requirements done for purpose of the evaluation are as follows

5.2.1 Adaptation for keeping a log of all updates to cells

No	Adaptation	Brief Description /Use
A4	Auto entry (date time) for each entry, i.e. whenever spreadsheet content is changed, a track of change is kept.	Works like a log for keeping information about the cell changes.

Table 5.3: Adaptation A4 keeping log of all cell updates.

5.2.1.1Description

Every time we make a change in a cell(s) of a Spreadsheet, we require to record the time and date (can be the current system time and date) to be recorded in a log file. Also one may like to add the username of the person who edits that cell(s).

5.2.1.2 Use

This can be a useful feature where an application is accessed and modified by many people and we want to know who did it and when that one did it.

Since our motivating example is a standalone application (we assume for this case study) that there are no concurrency issues to be considered like a cell being updated by two people at same time.

5.2.1.3 Adaptation Procedure

We instrument method *void setValue (String dest, String value)* from spreadsheet using the AIDE technology. Using CSL parser we get more semantics of this instrumented with the old semantics.

Then in *glue code* we write a method beforeInsert ()

This method consists of a mechanism that writes a record like structure in a log file for instance with the cell being updated, value, time, date.

```
/**
 * Returns the cell data value string ,before a data is inserted in
 * the spreadsheet keep its log.
 */
```

```
   public String beforeInsert(edu.wpi.cs.adapt.Context c) {
        Object args[] = c.getArgs();
        String dest = (String) args[0];//get dest
        String s = (String) args[1];//get value
        tempvalue =s;
         if (s != null) {
            Date tdate = new Date();//get date
            //create log string for given data
            String value = "Dest:" + dest + "," + "Data:" + s + "," + "
                                                       Date:" + tdate;
            //update logfile with data
            write("Spreadsheet.log", value, ";");
          }
         return s;//return input string value
    } //end of beforeinsert()
```

In CSL file we write

```
component adaptme adapts spreadsheet{

    class Glue;

    action beforeInsert(java.lang.String dest,java.lang.String
                                                          value);

    method void setValue (java.lang.String dest,java.lang.String
                                                          value){
            before beforeInsert(dest,value);
    };

};//end of component
```

In above CSL script we try to invoke beforeInsert ( ) before we set the value of the given cell in the spreadsheet and get following output.

5.2.1.4 Output

We can relate this output by mapping a small table like object that consist the destination address (one of the input parameter of setvalue () function), data and relevant user/time-date information associate with it.

Sample File: Spreadsheet. Log

```
Dest:D2,Data:http://www.wpi.edu/~sudeepu/abc.html,  Date:Tue  Apr
08  11:04:48  EDT  2003;Dest:D5,Data:sudeepu@wpi.edu,  Date:Tue  Apr
08 11:05:22 EDT 2003;Dest:D5,Data:, Date:Tue Apr 08 11:10:36 EDT
2003;Dest:B4,Data:sudeepu@wpi.edu,  Date:Tue  Apr  08  11:10:40  EDT
2003;Dest:B1,Data:,     Date:Tue     Apr     08     11:13:46     EDT
2003;Dest:B2,Data:D2,    Date:Tue    Apr    08    11:14:10    EDT
2003;Dest:C6,Data:B2,    Date:Tue    Apr    08    11:14:18    EDT
2003;Dest:D4,Data:sheet0:D2,   Date:Tue   Apr   08   11:14:47   EDT
2003;Dest:D6,Data:D4,    Date:Tue    Apr    08    11:15:07    EDT
2003;Dest:B5,Data:sheet0:D2,  Date:Tue  Apr  08  11:20:29  EDT  2003
<eof>
```

Note: *output shown above is actual output of the adaptation*

5.2.2 Adaptation for supporting multiple spreadsheets

No	Adaptation	Brief Description /Use
A5	Multiple Spreadsheets, ability to shift to different sheet based on an event. Also ability to do reference between the spreadsheets.	Like multiple sheets in excel or other commercial spreadsheets. Referencing here indicates one can refer to a particular spreadsheets value in other. e.g. sheet2:B2

Table 5.4: Adaptation A5 Ability to have multiple Spreadsheets.

5.2.2.1Description

Like commercially available spreadsheets, we adapt the application to have multiple sheets open and thus allowing us to use the data between the two sheets for computation purposes

Some requirements,

- We have to have multiple sheets
- We need to identify those individual sheets with unique names like e.g. Sheet5
- We should be able to address a particular sheet's cell(s) and get data (using reference) so as we can use it in other sheet calculation

5.2.2.2 Use

This multiple sheet adaptation is based on the feature of the commercially available spreadsheet based applications that allow similar kind of functionality. This functionality is especially effective when we can refer to the data from one sheet to the current sheet. This kind of ability in an application on practical basis adds a powerful functionality and commercial value.

Note: *This adaptation is important since it suggest how to adapt the container of application to be adapted since here we adapt the class FinalApp itself. This adaptation is very easy if we modify the FinalApp but doing it using our technique and writing relevant code in glue code may be bit difficult but demonstrates the flexibility and applicability of our approach.*

5.2.2.3 Adaptation Procedure

To have multiple instances of spreadsheet and we need spreadsheet objects. We do so by maintaining an array of spreadsheet objects in the Glue code. These sheets are assigned to the actual sheet in the original application on basis of an event and previous sheet is updated and saved back to the array. Sheets need a way to refer/communicate with each other (that is share the data). We now have spreadsheet objects distinguishably identified by a unique identifier still in memory (or can be loaded into memory on demand using persistent storage). We can access that data from a particular sheet by carrying out one more adaptation that checks the input string for the reference to the other sheet.

We can use the same GUI and shift between Spreadsheet beans using triggered by an event set by clicking the sheet button. The button is a customizable feature present in GUI. This button has dummy method forSheetChange() associated with its actionperformed event.

Here one can think about adapting the GUI to add the button using glue but we for sake of simplicity avoided that adaptation.

Here, we instrument method forSheetChange() from FinalApp, which basically is the method present to customize the actionperformed for a given customizable button in the GUI.

The total number of spreadsheet beans is customizable using a global variable in Glue code.

Then in *glue code* we write a method forSheetChange ()

This method consists of a mechanism that handles the changing of sheets from one sheet to other consistently.

```
/**
 *Takes care for multiple Sheets Option working,and
 *changing sheet once the button is clicked
 */
  public void forSheetChange(edu.wpi.cs.adapt.Context c) {
      Object args[] = c.getArgs();//context related
     if (count >= 0 && count <= (MULTIPLE_SHEETS-1)){
         // to keep  circular flow between sheets
         spr[count] = fa.spreadsheet;
         if (count == (MULTIPLE_SHEETS-1)){
            count = 0;
            //go back to first sheet after triggered sheet change
            //at last one
          } else {
              count++;
            //to maintain 1,2,3,n,1,2,.like change between sheets
          }
          fa.textBean.setText("TO NEXT SHEET" + count);
          //indicate which is current sheet
          fa.setspreadsheet(spr[count]);
          //assign selected sheet as current sheet
          if (initcount < count + 1) {
                initcount++;
               if(initcount == 2 &&
                            REFLECT_ADAPTATION_MULTIPLE_SHEETS ){

               //get adapter of originally adapted spreadsheet
               //and set if the same for other spreadheets
                       ca = fa.spreadsheet.getAdapter();
                }
                // Create invisible SpreadsheetBean
                fa.spreadsheet = new Spreadsheet();
                // set adapter for other spreadsheets so as they
                //inherit the original spreadsheet's existing
                // adaptations
                fa.spreadsheet.setAdapter(ca);
                // refresh events come in from the spreadsheet,
                fa.spreadsheet.addSpreadsheetListener(fa);
           } //end of if
           //reinitialize TableBean to show new spreadsheet info
           fa.tb.clearTableValue();
           fa.tb.repaint();
           fa.setSeo();
     }

    }//end of forSheetChange(Context)
```

In CSL file we write

```
application FinalApp {
implements      ActionListener ,      TableListener ;
component app   instanceof FinalApp;
component adaptfa adapts app{
    class Glue;
    action forSheetChange();
    method void doSheetButton(){
      before  forSheetChange();
    };
};//end of component

}
```

In above CSL script we try to invoke forSheetChange ( ) before doSheetButton() is executed.

For the Adaptation that enables data usage between the sheets using reference, we add callback to the method evaluateConstant() from Spreadsheet, which is already instrumented for the basic adaptations that we worked out.

Then in *glue code* we write a method beforeEvaluateSheet ()

This method consists of a mechanism that handles the referencing of cells to get data from one sheet to other consistently.

The code goes as follows

```
/**
 * Checks the string if in case of multiple sheet selection *
 * reference is made to other sheet
 * Note implementation will work only of sheets upto 0-99
 * Note implementation works only when the referenced cell has a
 * value in it & the refereced cell is not referencing other
 * cell (this condition is as yet not handled here)
 */
 public String beforeEvaluateSheet(edu.wpi.cs.adapt.Context c) {

    // String is the first argument
    Object args[] = c.getArgs();
    String s = (String) args[0];
    if (s.startsWith("sheet")) {
        int index = "sheet".length();
        //check for syntax if proper e.g. sheet0:A1
        String n = s.substring(index + 1, index + 2);
        String m = s.substring(index + 2, index + 3);
        if (m.equalsIgnoreCase(":")) {
            // get index of the sheet and address of csl
            int sheet_index = Integer.parseInt(s.substring(index,
                                                 index + 2), 10);
            //get value for that particular sheet
            s = spr[sheet_index].getValue(s.substring(index +
                                                           3));
            // append the reflected value in current sheet
            args[0] = s;
            return s;
         } else
            if (n.equalsIgnoreCase(":")) {
                // get index of the sheet and address of csl
                int sheet_index = Integer.parseInt(s.substring(
                                        index, index + 1), 10);
                //get value for that particular sheet
                s= spr[sheet_index].getValue(s.substring(index +
                                                           2));
                // append the reflected value in current sheet
                args[0] = s;
                return s;
            }
    }
    return s;

} // end of beforeEvaluateSheet(Context)
```

In CSL file we write

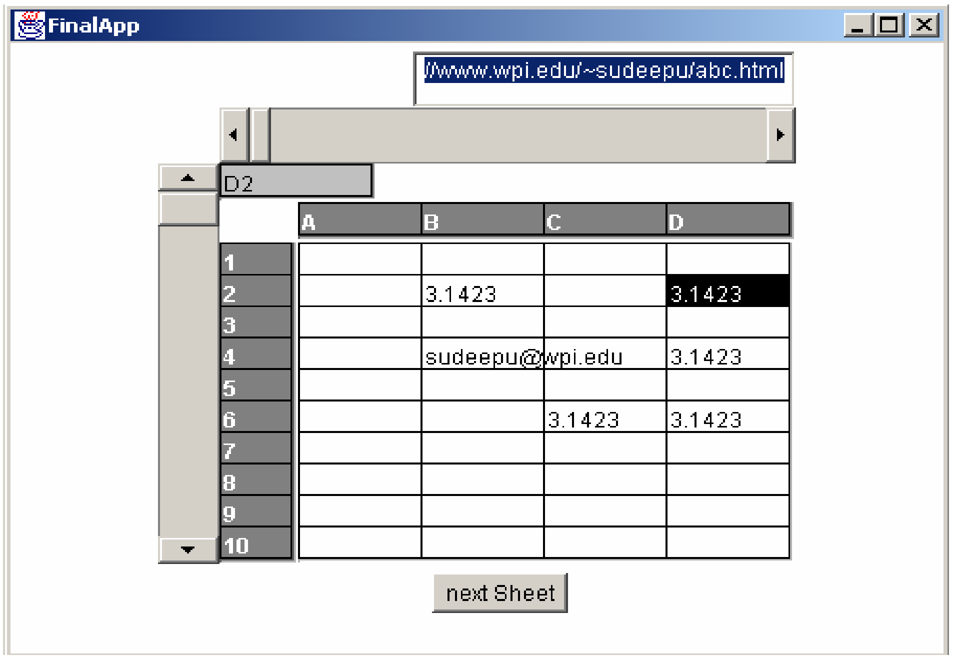
```
component adaptme adapts spreadsheet{

    class Glue;

    action beforeEvaluateSheet(java.lang.String s);

    method float evaluateConstant (java.lang.String s){
      before beforeEvaluateSheet(s);
    };

};//end of component
```

In above CSL script we try to invoke beforeEvaluateSheet ( ) before evaluateConstant() is executed.

5.2.2.4 Output

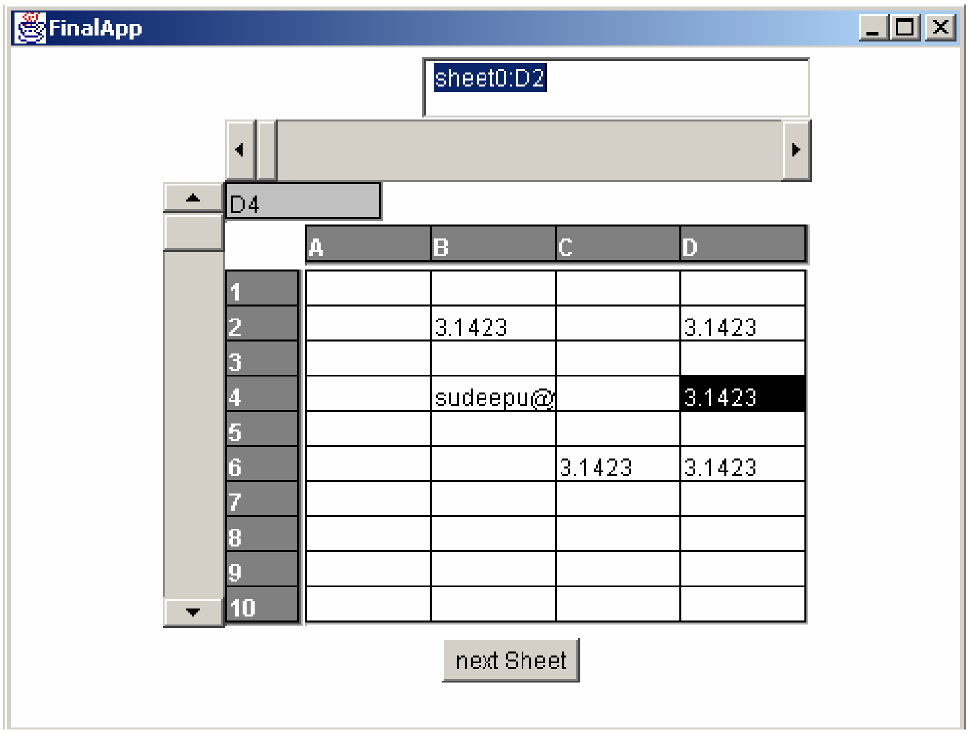
We will go through few screen shots, which show the behavior of the application when the above adaptation is done.

Consider following spreadsheet sheet 0, where in a number is inputted from the URL, which is our existing basic adaptation. In figure 5.1, figure 5.2 and figure 5.3, we can see how referencing works within a single sheet. Figure 5.2 demonstrates that our adaptation for the inter-sheet referencing gives the expected result as that of the actual intra-sheet reference handled in Spreadsheet class.



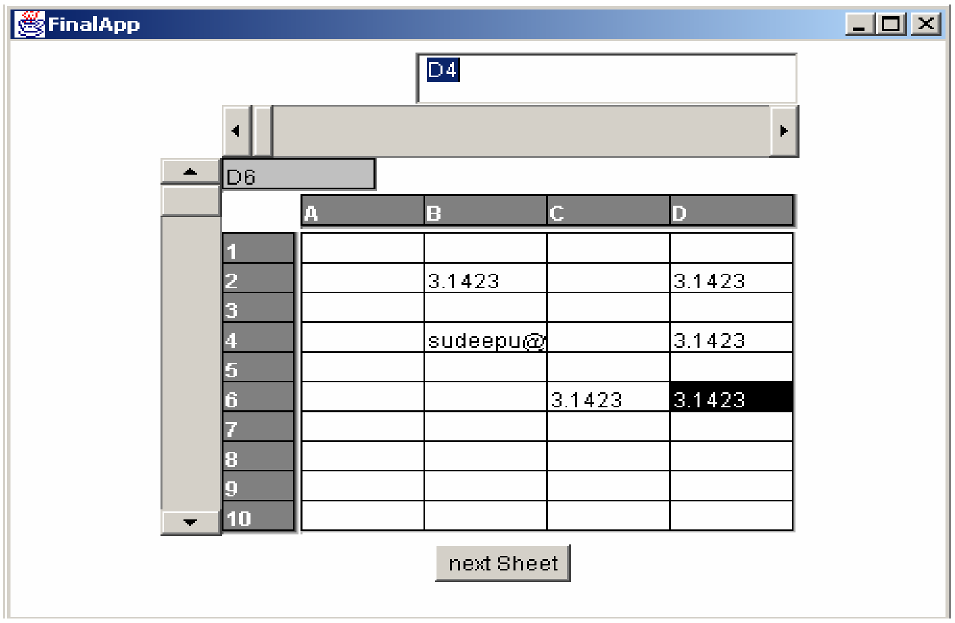
	A	B	C	D
1				
2		3.1423		3.1423
3				
4		sudeepu@wpi.edu		3.1423
5				
6			3.1423	3.1423
7				
8				
9				
10				

Figure 5.1: intra-Sheet Referencing.



	A	B	C	D
1				
2		3.1423		3.1423
3				
4		sudeepu@		3.1423
5				
6			3.1423	3.1423
7				
8				
9				
10				

Figure 5.2: Intra-Sheet Referencing using Adaptation.

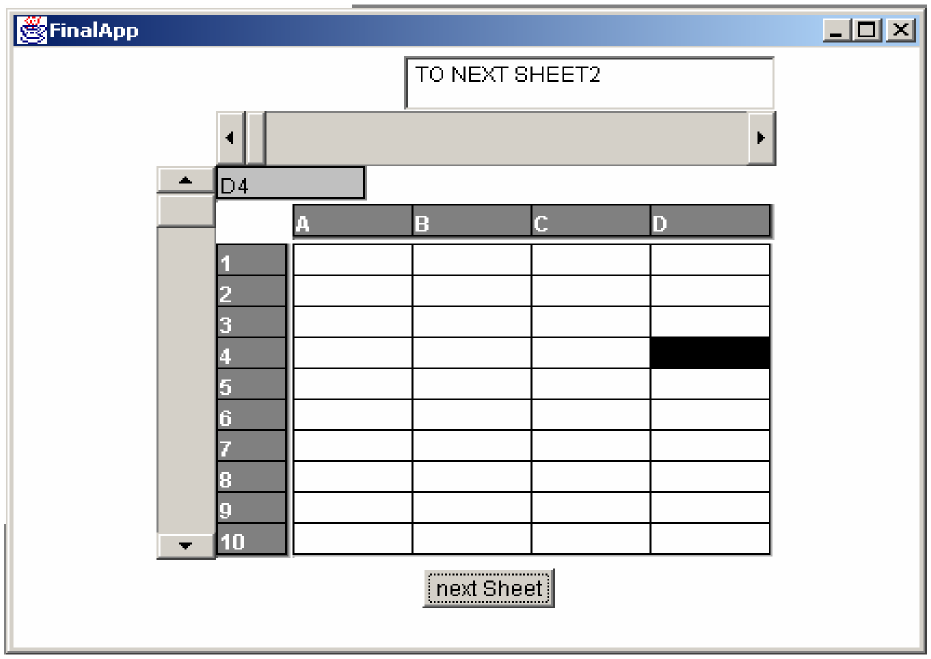


D4

D6

	A	B	C	D
1				
2		3.1423		3.1423
3				
4		sudeepu@		3.1423
5				
6			3.1423	3.1423
7				
8				
9				
10				

Figure 5.3: Intra-Sheet Referencing using actual Spreadsheet Class.

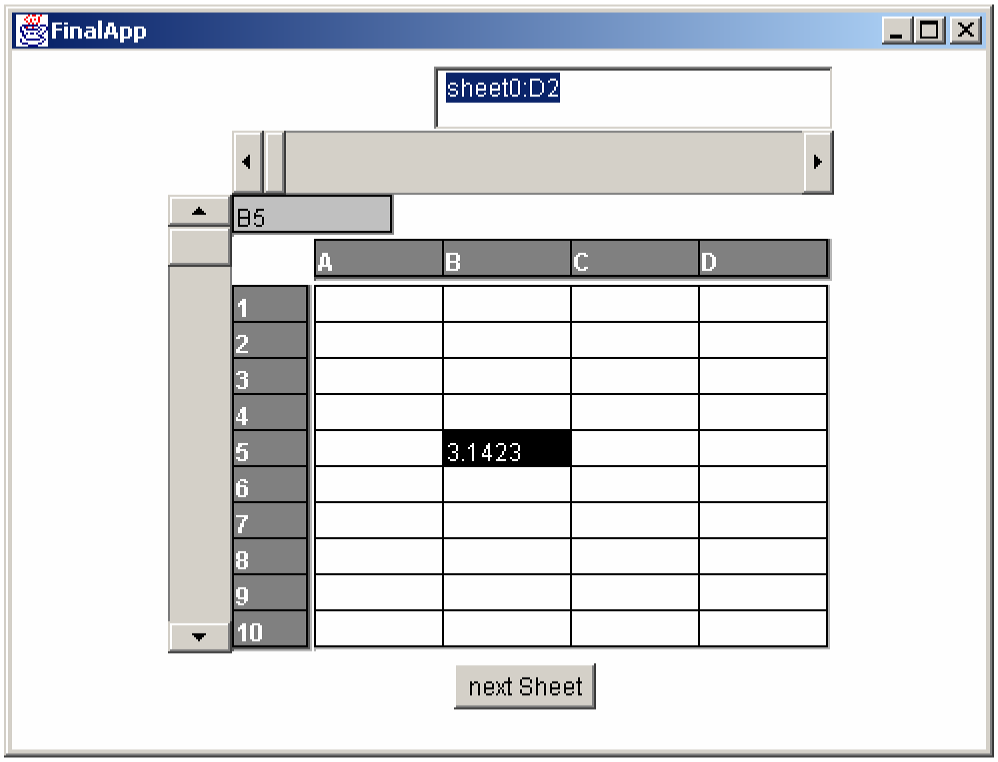


TO NEXT SHEET2

D4

	A	B	C	D
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

Figure 5.4: Changing Sheets Adaptation using next sheet Button.



	A	B	C	D
1				
2				
3				
4				
5		3.1423		
6				
7				
8				
9				
10				

Figure 5.5: Intra-Sheet Referencing using Adaptation.

Figures 5.4 and 5.5 shows how to change the sheet and how to use the inter-sheet reference between two sheets for the data.

This adaptation is still in the basic reference stage; the aspect that a cell can refer to another cell's value is done. When the other cell's value is again a reference to different cell in the sheet it simply returns the reference instead of going ahead and getting that reference's value. This can be handled by adding more intelligent code to the glue section of this adaptation. We did not handle this in given adaptation as the goal was just to show that we could refer to data from other sheet.

5.2.3 Adaptation for detecting if the input is email address and adding it to address book

No	Adaptation	Brief Description /Use
A6	Scan Input of textbox for email address, if valid email address say add to the address book	Filter like adaptation to do some meaningful operation on the inputs to the spreadsheet.

Table 5.5: Adaptation A6 keeping log of all email addresses.

5.2.3.1Description

Every time a valid email address is input in the spreadsheet, it should be entered in to an address book. (Where an address book can be a different component of a target system)

5.2.3.2 Use

This email validation adaptation is useful to demonstrate that we can scan an input for useful data and reuse the data for some other purpose, which is external to the application being adapted.

In this case, we are scanning for a data if it is a valid email address and if found so we add this data to an address book.

Note: *This adaptation is important since it suggest how to incorporate external application in the glue code to do an specific task .We incorporate a JavaCC based email format parser which parses input data for popular email format based on the grammar specified. This demonstrates the flexibility and applicability of our approach for doing more complex adaptations that might involve inclusion of third party components for adaptations. It also thus suggests the deployment of such components/applications used in adaptation.*

5.2.3.3 Adaptation Procedure

To check if input data is email or not we invoke the JavaCC based Email parser giving the input data as the data to be parsed. If this parser returns a valid parse we conclude that the input data is a valid email address and we append it in the corresponding email address book file.We do not alter the input to spreadsheet as in case of URL adaptation, but we are using it for external purpose like keeping track for all email addresses entered in the spreadsheet.

We add callback to the method evaluateConstant() from Spreadsheet, which is already instrumented for the basic adaptations that we worked out.

Then in *glue code* we write a method beforeEvaluateMail ()

```
/**
 * Returns the same string itself,before that it evaluates if
string
 * is a valid EMAIL Address, if valid email address stores it in
file
*/
 public String beforeEvaluateMail(edu.wpi.cs.adapt.Context c) {
        Object args[] = c.getArgs();
        String s = (String) args[0];
        boolean flag = false; //set flag to false
        String value; //value of string
       if (s != null) {
            String args1[] = new String[1];
            args1[0] = s;
            // invoke the parser to check email format
            // pass the string as arguments
            EMAIL.main(args1);
            flag = EMAIL.checkEmailFormat();
          //flag is true if valid email format
        }
            // if valid email format
            if (flag) {
                value = s;// valid email address: value
                v.add(value);
               //store the address in emailLog.log file
                write("emailLog.log", s, ";");
            }
            flag = false;//reset flag
            return s;//return the original string

  } //end of beforeEvaluateMail()
```

The method beforeEvaluateMail () takes a String as an input and parses the string to check if it is an email address. If a match is found it stores the address in an address book (output file) and the string reflected in the spreadsheet would be same as it was inputted this method should not in any way change this string (as in case of URL Adaptation).

Note*: here valid email address parsed is of grammar*

<name> <@ >(<name> <.> <name>)+

For example, aa@ ss, aa@ss.com are valid email formats assumed for the adaptation.

In CSL file we write

```
component adaptme adapts spreadsheet{

    class Glue;

    action beforeEvaluateMail (java.lang.String s );

    method float evaluateConstant (java.lang.String s){
       before beforeEvaluateMail(s);
    };

 };//end of component
```

In above CSL script we try to invoke beforeEvaluateMail ( ) before evaluateConstant() is executed.

5.2.3.4 Output

The screen shot in Figure 5.6 shows the behavior of the application when the above adaptation is done.

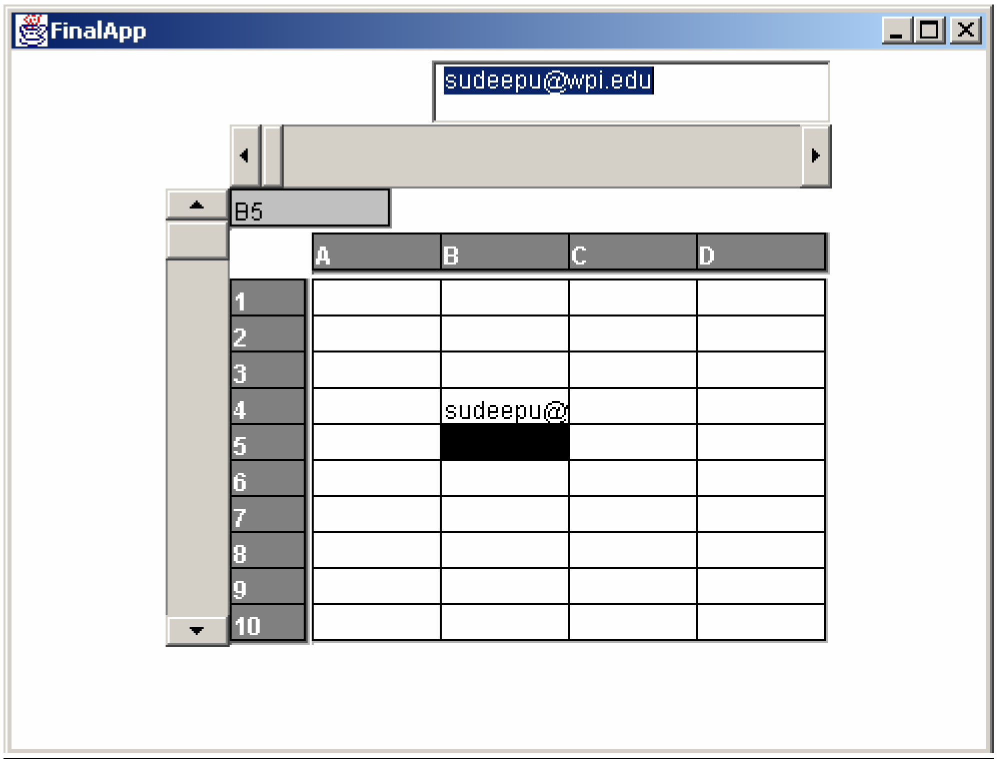


Figure 5.6: Email Adaptation.

Sample File: emailLog. Log

```
sss@hotmail.com;sudeepu@hotmail.com;sss@ss;a@bcom;s@s.com;aa@ss.c
om;aa@ss.com;aa@wpi.edu;aa@b.com;cc@d.com;ee@g.com;aa@ss.com;bb@s
s.com;aa@ss.com;a@a;sudeepu@wpi.edu;sudeepu@wpi.edu
```

Note*: output shown above is actual output of the adaptation*

5.2.4 Adaptation for detecting if the input is a valid picture (a path) display picture

No	Adaptation	Brief Description /Use
A7	Adapt spreadsheet to display a picture in a given cell.	If input string is a jpeg, gif file with appropriate path/reference, display the picture in the TableBean tb

Table 5.6: Adaptation A7 display picture in Spreadsheet.

5.2.4.1Description

Every time we insert a valid picture (jpg, gif …depends on the supporting glue code implementation) path into a cell it should be shown as a images in the spreadsheet. This is a massive adaptation of the spreadsheet, which is not practically designed to show a picture, thus this adaptation equally challenging and complex to do.

5.2.4.2 Use

This image adaptation is useful to demonstrate that we can change the basic behavior (in some case can be rightfully called as functionality) of the application being adapted. The spreadsheet is not designed to show images. We are here trying to force the spreadsheet application to do so. This approach signifies the importance of how we can use our proposed technique to alter functional behavior of a given application to be adapted.

<u>Note</u>: *This adaptation is important since it suggest how to adapt the internal components of a given application. For instance we are here adapting TableBean class, which is an internal component of the GUI of this application. This again demonstrates the reach and applicability of our approach for complex adaptations involving internal components of the given application (which we assume has Component Based Architecture).*

5.2.4.3 Adaptation Procedure

We instrument method redraw() (TableBean class), which is invoked everytime there is a change in TableBean.

We add callback to the method redraw() from TableBean. In glue code afterEvaluateGraphics() method check s the input if it's a valid image path. If valid, it retrives the image for the given path in form of java.awt.image . This image is drawn in table bean by calling method Graphics.drawImage() using appropriate coordinates .

Then in *glue code* we write a method afterEvaluateGraphics()

```
/**
 * before a data is inserted in the spreadsheet,evaluates as
String
 * if it is image and draws it in tablebean.
 */
 public void afterEvaluateGraphics(edu.wpi.cs.adapt.Context c) {
       // Graphics  is the first argument
        Object args[] = c.getArgs();
        java.awt.Graphics g = (java.awt.Graphics) args[0];
        //get Graphics Object g
        int col = ((Integer) args[1]).intValue() ;//get column
        int row = ((Integer) args[2]).intValue() ;//get row
        int TEXT_XPAD = 2;//set properties x
        int TEXT_YPAD = 2;//set properties y
        int columnWidth =fa.tb.getColumnWidth();//set columnWidth
        int rowHeight =fa.tb.getRowHeight();//set rowHeight
        int topCorner = (columnWidth + TEXT_XPAD) * (col - 1);
        tempvalue=fa.tb.getTableValue(col,row);//get value of
string

        if(tempvalue!=null){//if not null
            //check if string is valid image here gif,jpg
            ImageIcon icon = new ImageIcon(tempvalue);
            Image Im = icon.getImage();
            ImageObserver IO = icon.getImageObserver();
            //draw image
            g.drawImage(Im,TEXT_XPAD + topCorner,(rowHeight +
                             TEXT_YPAD) * row - TEXT_YPAD,IO);
        }//end of if

    }//end of afterEvaluateGraphics(Context)
```

In CSL file we write

```
// tb is to be defined as instance of TableBean in main
application CSL script

component adapttb adapts tb{
    class Glue;
    action afterEvaluateGraphics(java.awt.Graphics g, int
                                    viewCol, int viewRow);
    method void redraw (java.awt.Graphics g, int viewCol, int
                                                 viewRow){
       after  afterEvaluateGraphics( g, viewCol,viewRow);
    };
 };//end of component
```

In above CSL script we try to invoke afterEvaluateGraphics() after

redraw() method is executed.

5.2.3.4 Output

Figure 5.7 shows the behavior of the application for above adaptation.

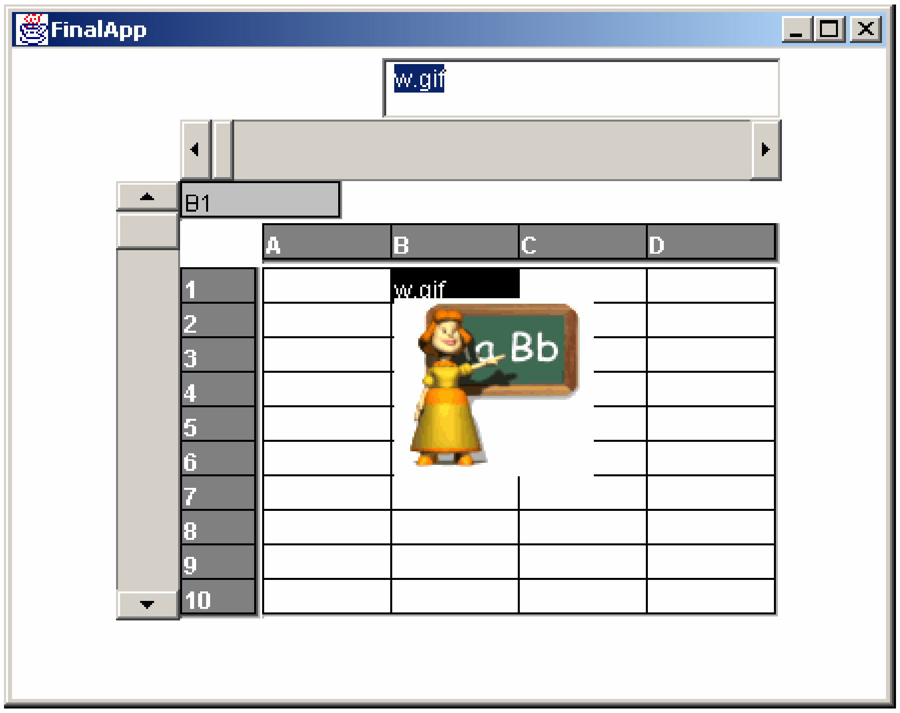


Figure 5.7: Picture Adaptation.

5.2.4 All Adaptation together

5.2.4.1Description

It is important that we have all the adaptations that we described so far in this thesis coexist.

5.2.4.2 Analysis

It is very important that we do the adaptations such that they should coexist and be valid, all at same time. There were instances that two adaptations could be contradictory and thus one has to be very careful that when the adaptations are chose, the interdependencies between these adaptations are taken in to account.

It is quite possible that adaptations may be affecting the state of application undesirable to the other adaptation. Few of the factors that may lead to unwanted adaptation result are listed in section 5.5 under shortcomings. Also there can be an issue way in which adaptations are done. For instance we have three adaptations URL, Email and referencing between multiple sheets which are to be done before the same method. The order in which adaptations are done does matter. For example if I do URL adaptation before email adaptation, The URL adaptation changes the state of input to actual evaluateConstant()

method .This implies that the email adaptation would evaluate the output of URL for valid email address instead of actual input by the user .So the order of adaptation is important and we leave the liberty of deciding this factor to the application assembler who carries out these adaptations by appropriately updating the CSL script entries.

5.2.4.3 Output

We have been successful in coexisting all the adaptations presented in this thesis with respect to our spreadsheet example. The concerned CSL scripts are presented in the appendix E section of this thesis.

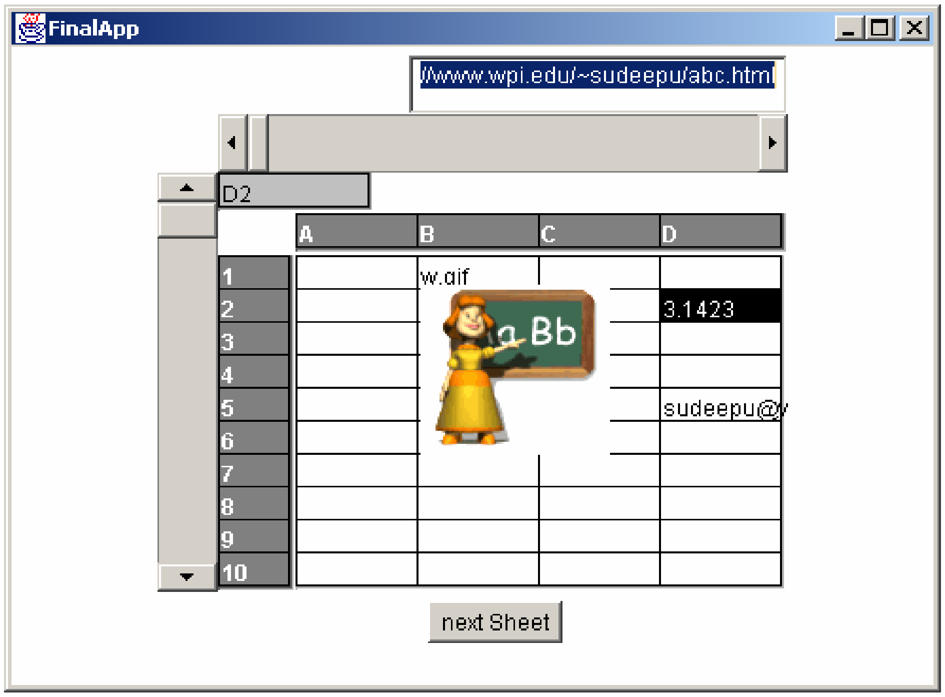


Figure 5.8: All Adaptations.

The screen shot in figure 5.8 shows the application when the all adaptations are in effect.

5.3 ADAPTATION IN CMI DEPLOYMENT

5.3.1Description

In most cases, we can see a component deployment strategy being used for software systems .We saw a need for integrating our approach with a popular component deployment technique. We selected CMI [1], which is one of component deployment strategies as EJB [20] and CORBA [21].

We continue further from the discussion done in deployment Section 4.6 in Chapter 4 and show how to integrate our approach with the CMI deployment.

5.3.2 CMI: Background, Deployment & Use

This adaptation suggests how we can adapt components that form a software system and show that they can be deployed in the similar manner, as they would have been without adaptation, using little effort.

In CMI, we deploy few components that form a software system for taking put of two numbers and doing operations on it. These operations are defined in a Class called Calculator. CMI requires scripts that give it a sequence and description of components to be instantiated, activated, connected and deactivated.

CMI controls the life cycle of the application by reading these scripts and appropriately doing instantiation, connection, and activation and finally deactivation of the components. CMI connects the components by matching their required and provided interfaces. It is mandatory for each component to implement IBlock interface.

CMI controls the whole procedure through a container called "Foundation". We need to make changes to this Foundation, which is main CMI container. The important changes would be a mechanism to add appropriate callbacks to the adapted component and registration of GlueCalc (our glue code) methods with the concerned adaptable component methods. This also implies need for the container to have mechanism that invokes the CSL parser and gets the information from the CSL files. The information about the glue code is given in section 5.3.3 and the concerned CSL scripts are provided in Appendix E.

As shown in Figure 5.10 each component block has its own CSL script that suggests information about that component and its adaptable elements. Even the new component that adapts the old component has its own csl describing the adaptation to be imposed (which is Modified Component in our case).

Figure 5.9 gives us an overview of deployment of this example of Calculator that is similar to deployment we discussed in Section 4.6 in Chapter 4.

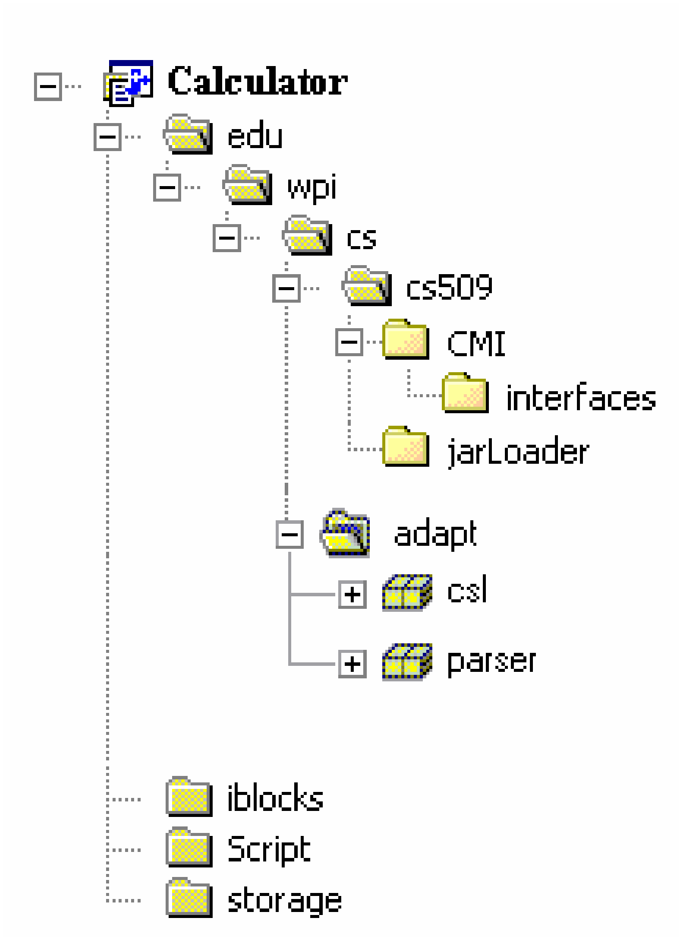


Figure 5.9: Deployment in CMI.

Here the package edu.wpi.cs.adapt is required along with the packages that enable parsing and CSL structures. The scripts are updated with the new component (Modified), which adapts an old component (Calculator in this case). The component to be adapted is to me an adaptable component again which is possible by using AIDE. A Block hence contains the component with its concerned CSL script, which now the updated container “Foundation” is capable of reading.

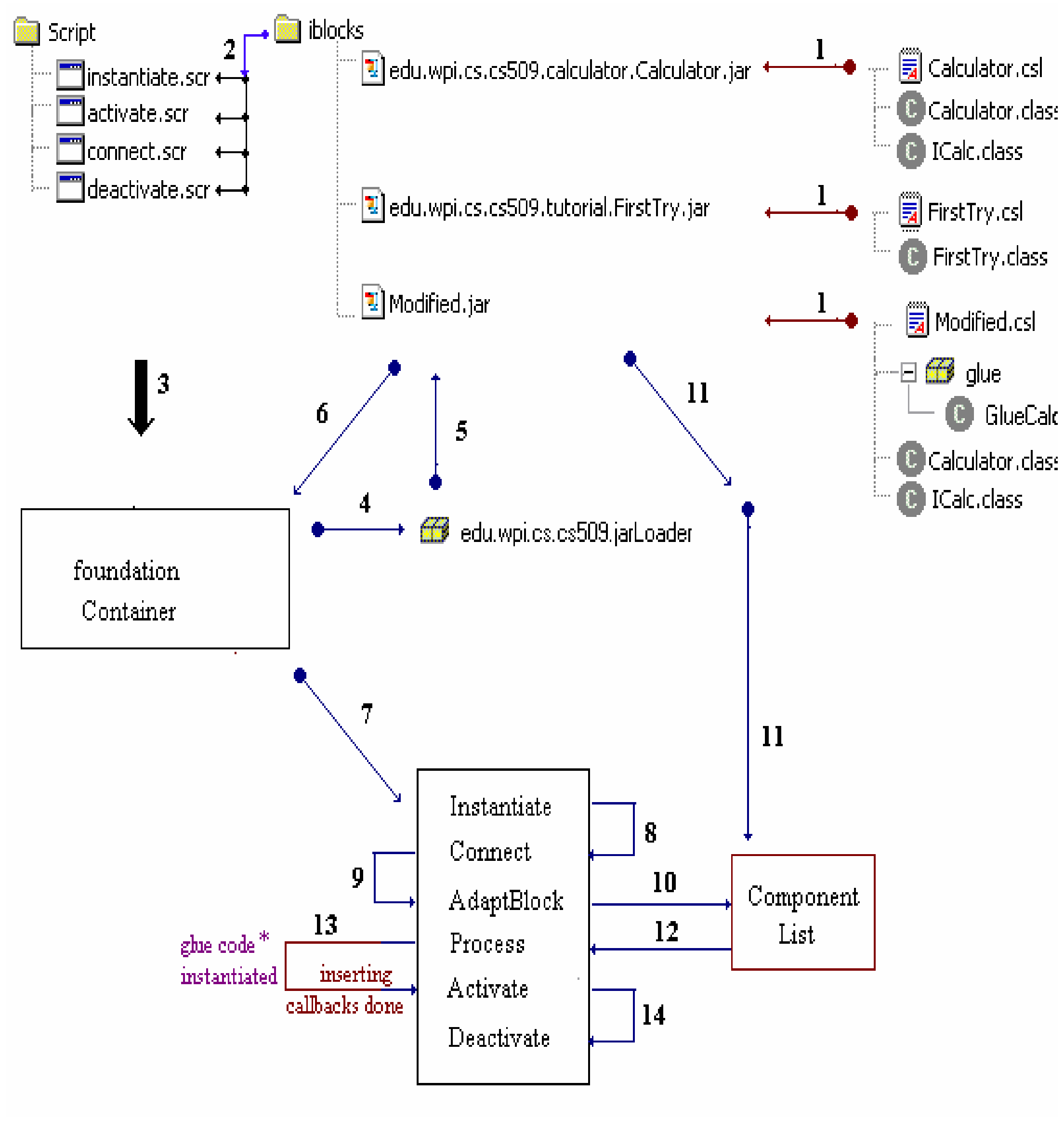


Figure 5.10: Integration of adaptation technology and CMI.

Figure 5.10 can be briefly explained in following steps

1. Construct the blocks with proper components and their CSL script files. The adaptable components must be updated. The components adapting other components need to be packaged with the CSL script that suggests the adaptations to be imposed.
2. The block entries are to be updated appropriately in the script files.
3. Foundation parses the scripts and gets the block list to be loaded.
4. Foundation calls jar loader to load the blocks.
5. Jar loader searches for the blocks with given names.
6. Jar loader loads the required blocks.
7. Foundation instantiates the blocks in the sequence specified in the instantiation script.
8. Foundation connects the blocks on the specification in the connection script and also checking the provided and required interfaces of blocks to be connected.
9. Foundation calls method adaptBlock to check for components having their CSL specification deployed with them.
10. adaptBlock goes through the block (component) list passed by foundation.
11. adaptBlock seek the CSL script files if present for each block.
12. These CSL files are passed to the process method.

13. Process method, parses the CSL specification, instantiates GlueCalc and inserts callbacks on the adapter of the adaptable block (Calculator component here).
14. All blocks are activated and application runs, Finally application calls deactivate to shutdown cleanly.

Note: *This adaptation is important since it suggest how to adapt the internal components of a given software system using a specific deployment strategy. This example can be used as a template to carry out adaptations for black box component based systems. This again demonstrates the capability and applicability of our approach for complex adaptations involving integration with component deployment techniques.*

5.3.3 Adaptation Procedure

We instrument method multiply() (Calculator class), which is invoked when multiplication operation is selected. We add callback to the method multiply() from Foundation. In GlueCalc code beforeMultiply() method increases the value of inputs by 10.

Then in GlueCalc we write a method beforeMultiply()

```
/**
 *  Increases the value of numbers inputted by 10
 *  @see edu.wpi.cs.cs509.calculator.Calculator
 */
 public void beforeMultiply(edu.wpi.cs.adapt.Context c) {
    // int is the first argument
    Object args[] = c.getArgs();
    try {
          int a = ((Integer) args[0]).intValue();
          int b = ((Integer) args[1]).intValue();
          //increase the input by 10
          // one can do more complex adaptations here following
          //is just an simple example to
          // illustrate how one can do it technically.
            a = a + 10;
            b = b + 10;
          //assign the value back to Adapter so as the new
          //values are reflected in the actual routine
            args[0] = new Integer(a);
            args[1] = new Integer(b);
     } catch (Exception e) {
           //handle exception
     }
} //end of beforeMultiply()
```

In CSL file we write

```
component Modified adapts edu.wpi.cs.cs509.calculator.Calculator
{
    class glue.GlueCalc;
    action beforeMultiply(int a ,int b);
    // apply the adaptation to the multiply method

     method int multiply (int a ,int b) {
        before beforeMultiply (int a ,int b);
     };
};
```

In above CSL script we try to invoke beforeMultiply() before multiply() method is executed.

5.3.3.4 Output

Appendix D shows the output of the CMI run

5.4 COMPOSABLITY OF ADAPTATION TECHNIQUE WITH RESPECT TO ADAPTATION

Adaptation technique should be composable with the component it is applied on and should also be composable to the other components as it was before adaptation.

Our approach is composable as in case shown in Figure 5.11

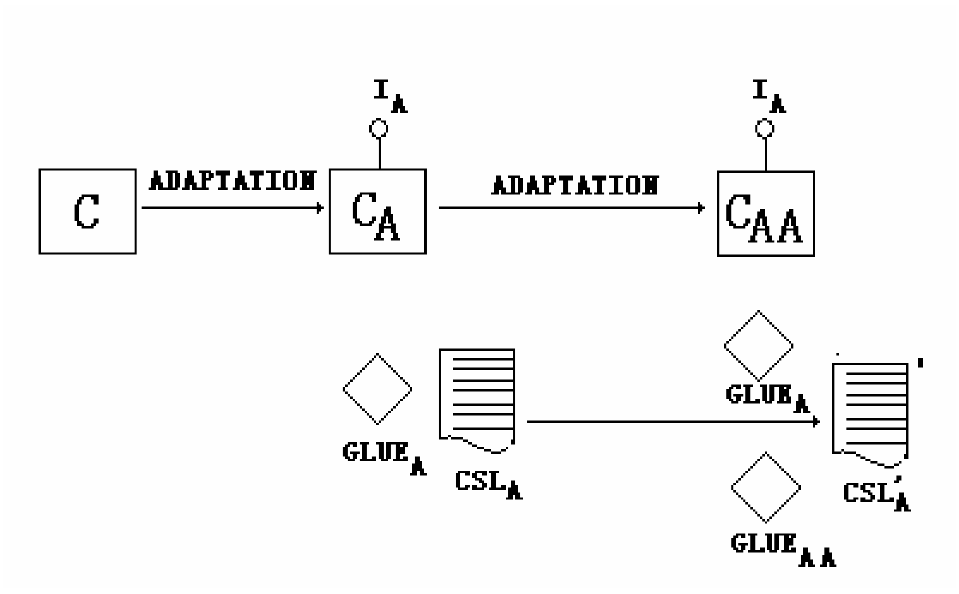


Figure 5.11: Composablity approach.

In Figure 5.11, we can have a component C adapted to component C_A. This component C_A now has its own glue code $GLUE_A$ and its own CSL script CSL_A. We propose to compose this adaptation by carrying out adaptation of this adapted component C_A such that component C_{AA} is obtained, where all adaptations are valid as that in case of C_A as well as the new adaptation. This Composablity is obtained by editing the CSL specification that was present after adaptation of C. Editing the CSL script CSL_A we can easily control and impose new adaptation. For example, consider that we adapt C with method x, such that method beforeX from $GLUE_A$ is registered with a before callback. We thus obtain C_A. We need adapt the component C_A with method x such that beforeXX method in $GLUE_{AA}$ is registered with before callback. Editing CSL_A to CSL_A' gives us flexibility of deciding and imposing if method beforeX has to be registered before the method beforeXX. This process involves adding few lines to the existing CSL_A and in desired sequence. No extra functionality is required for container to know about the glue code and adaptations of the composed component C_A that is being adapted to derive new adapted component C_{AA}. We are thus, successful in evaluating the Composablity metric.

In Figure 5.12, we propose one more approach to achieve Composablity. We can get the adapted component C_A from C as described above. In this approach, the main difference is that we now have one glue code at the C_A adaptation level which is $GLUE_{AA}$ .It becomes important that the new composed component C_{AA} is aware about the component C_A 's composablity. Here it is also imperative for C_{AA} to have knowledge about CSL_A and that the reference to the CSL_A from CSL_{AA} is properly handled. This implies need for an intelligent container, which would have the knowledge about the composablity of C_A and mechanism to parse the CSL_A (that is embedded with C_A) and register callback with respect to $GLUE_A$.

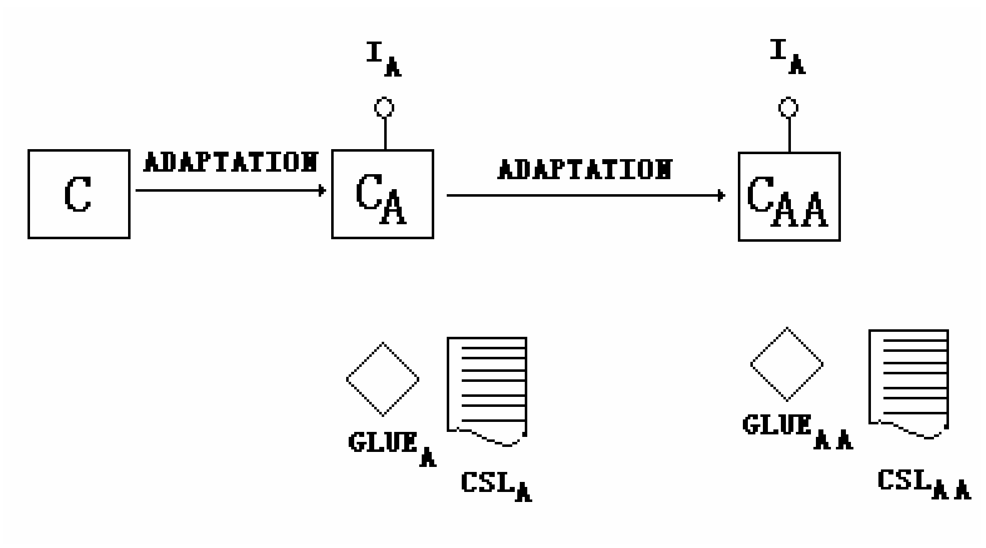


Figure 5.12: Composablity proposed approach.

The container would also be expected to do the actual work of parsing CSL_{AA} and registering callbacks with respect to $GLUE_{AA}$. This approach is similar to wrapping except the interface here I_A remains unchanged through out unlike wrapping.

Note: Here we also maintain composablity with respect to the other components, as the interfaces remain unaltered unlike the case of wrapping technique. Since interfaces are not changed the new adapted component is as composable as the original one. Example CMI adaptation discussed in this chapter.

5.5 EVALUATION WITH RESPECT TO METRICS PROPOSED

Table 5.1 gives the metrics to be considering for evaluation. Table 5.8 gives few observations with respect to those metrics

Metric	Metric evaluation with respect to Adaptation
Transparency	We can see this measure on basis of the example of CMI adaptation. From application assembler's point of view there is no major change in the actual component deployment procedure. One can disable the adaptations using CSL scripts and the whole CMI system works as if there was no adaptation. Even if the component is adapted no change is required in Client code. For example, in CMI, the component adapted does not require other client components to be changed for either functionality or connectivity. This ensures the transparency of the system.
Blackbox	We do not change the internals manually for enabling the adaptations. Yes, we do need the component to be adaptable which we achieve by using technology like AIDE. This ensures that the application assembler does not require code level access. The black box property thus is preserved.

Composablity	Composablity is to be done with respect to the component itself as well as with the components this adapted component interacts with. On basis of the discussion made in Section 5.4, our approach does support composablity.
Performance	This metric required us to use other available approaches for the same adaptation and evaluate the performances. We were unable to derive to the comparative performance metrics between our approach and other approaches. We were pleased to see that the performance was not much different when we took the perspective with respect to performance before adaptation and performance after adaptation.
Deployment	We have provided deployment examples indicating the integration of our approach with the deployment techniques like CMI. This ensures that the approach we propose can be deployed easily.

Table 5.7: Metrics evaluation with respect to adaptations

5.6 SHORTCOMINGS

We also were able to see few shortcomings of the approach, for example the kind of adaptations, which may not be possible or not feasible due to complexities as,

- Requirement of either writing complex and large glue code
- Unavailability of source code to instrument required method for adaptation
- Implementation of inter-state affecting adaptations on same method like for e.g. one adaptation adds a value v1 to the input and another adaptation adds value v2 to the input, if both of these adaptations are carried out on same method, the desired result of each adaptation is not achieved; however, only the *Application Assembler* knows if the group of adaptations are consistent.
- Implementation of contradictory adaptations on same method, for example, one adaptation sets input value v to *true* and another adaptation sets input value v to *false*. If both of these adaptations are carried out on same method, they both contradict and one does not get desired result.
- The approach is not well suited towards adapting user interfaces.
- If the members, we want to adapt have generalized inputs whose runtime type is not known in advance, then adaptation for that member can go wrong. For example, consider a method X having input of type Object. We, using any of the information from CSL

script or other resource for a black box component, cannot determine what kind of value or type the input may be during runtime. In such cases carrying out adaptation may not give desired result as this method is a bad candidate for adaptation

The above listed shortcomings are some of practical problems that can be faced by the application designers while using the framework we are supporting. There are many more such situations, which we have not mentioned as they can be of different nature and kinds depending on the application to be adapted, deployment, and many other factors.

We don’t claim that this framework we are presenting is appropriate for all adaptation needs. We presented a way of carrying out the adaptation as compared with the existing black box adaptation techniques and proposed that this technique would be better in some situations over the existing approaches.

5.7 SUMMARY OF CHAPTER

We proposed several metrics to evaluate this approach in Table 5.1. The candidates for evaluating this approach were the adaptations discussed in Section 5.2. We were also able to implement and evaluate our infrastructure with a component deployment technology CMI as specified in Section 5.3

In Section 5.5, we contrast the metrics proposed against the candidate adaptations selected and imposed for evaluation purposes and end this chapter with a brief note on shortcomings of this approach (Section 5.6).

Every adaptation method has its own shortcoming, though it depends on the kind of adaptation expected for the given blackbox component that will determine if our framework is better suited for a particular problem than other available technologies.

Chapter 6

CONCLUSION & FUTURE WORK

6.1 CONCLUSION

The goal of this thesis was to present an environment for specifying and executing adaptable software components. We proposed and defined the Component Specification Language grammar. We demonstrated, how AIDE [10] is integrated to assist in conversion of a component to an adaptable component. We also demonstrated, how to generate CSL specification for the given component using tools like CSLProfiler [15]. We provided parser for CSL specification (Chapter 2) and also data structures to appropriately hold the csl information.

We also presented several adaptations (Chapter2, 5) to demonstrate the flexibility of our approach in adapting a blackbox component (Chapter 2). We also demonstrated how to integrate our adaptation approach (Chapter 4) with the existing component deployment strategies like CMI (Chapter 5). The infrastructure addresses the problem of effective packaging of the components for third party use, adaptation, and deployment. Thus we support both component designer and component assembler.

Finally the bottom line is, in this thesis, we show how to use declarative specification to change run time behavior of a CBSE system.

6.2 FUTURE WORK

Future Work for this adaptation technique is primarily to make it more sophisticated and applicable on larger scale Following are areas where we visualize the future work to be done,

6.2.1 Automatic Glue code generation

As presented in this thesis, we manually edit and write the glue code. We also accordingly reflect the changes in the CSL script suggesting the adaptations to be imposed. We feel this can be made automated to further assist the framework. The visualized automation is that, once the candidate member of adaptable component is known and the type of callback is known, we can use a tool to generate the mechanical areas of glue code and reflect the corresponding changes in the related CSL script. Doing this will minimize the effort required to write a glue code where the application assembler would now have to just write few lines of code, that reflect the actual adaptations, in the skeleton of the glue code method.

6.2.2 Additional functionality in “parser for csl script”

As presented in this thesis, the existing parse does do a good job but there are still some issues that can make it more generic. For example, if consider in CSL script of component A, we have,

```
Component  A adapts componentB
```

Further in component B's CSL script, we have,

Component B adapts componentA

This will create a chain that might make the parser to go in an unwanted state. Current parser ignores this chaining after a certain level. But effort can be made to improvise the parser to deal with chaining problems

6.2.3 Developing better case studies & evaluating the adaptation technique for the same.

We present some case studies that demonstrate a variety of adaptations that we can do using our approach. But we feel strong need for development of better and different case studies. The adaptation of any component or application is very specific to factors like the way it is implemented, deployed and many other things. This makes each case study important provided that it involves a variety, which will demonstrate the uncovered features of this adaptation technique.

6.2.4 Attempting complex adaptations such as user interface.

We can attempt to do more complex adaptations like try change the runtime behavior for software system. For example, adapting a user interface using our approach is a challenge; it is a complex adaptation and may require more complex glue code. Achieving such adaptations help to evaluate the effectiveness of this approach to aid application assembler's decision in selecting the adaptation technique, if required.

6.2.5 adding more grammar to CSL to describe a blackbox component.

Attempt can be made to add more grammar to Component Specification Language that will lead to better and detailed understanding of a given black box component. This may lead to extended use out of the framework and approach we are suggesting.

6.2.6 Applying adaptation approach to other Object /structure oriented technologies

We strongly feel the need to extend this adaptation approach to other technologies like C++ etc. This will show the applicability of this approach and also open new avenues for the adaptations of blackbox components developed using other structured technologies like C++.

6.2.7 Adaptation of unstructured components

Section 6.2.6 leads to another work of extending this adaptation approach to the black box components developed using no structured technologies like C. This work would be very complex and may lead to rework of few areas. There also may be some new issues we didn’t come across due to the non-structural nature of the technology used.

6.2.8 Automating process of making an adaptable component and generating its csl like example integrating tools like AIDE and CSLprofiler.
We currently have to make a component adaptable using AIDE and either write down the components CSL script or generate it using tool like CSLProfiler. An attempt can be made towards development of an integrated tool that enables us to automatically instrument the given component and generate its corresponding CSL specification. Such tool will help application designer to package the adaptable component with its CSL specification.

6.2.9 Creating a standard library /platform to enable assembler to choose component deployment techniques.
Attempt can be made to develop a standard library to aid the different component deployment techniques. This will assist the component assembler to deploy the adaptable components using preferred component deployment strategy. This will minimize the manual process involved in the whole component adaptation process.

Most of future work suggested can be done towards making this adaptation approach more robust, automated and applicable.

APPENDIX A

Output of **AIDE** tool for Spreadsheet component:

```
<?xml version="1.0" encoding="UTF-8"?>
<class name="adapt.spreadsheet.Spreadsheet">
  <implements>SpreadsheetListener</implements>
  <implements>Serializable</implements>
  <extends>Object</extends>
  <modifier>public</modifier>
  <modifier>private</modifier>
  <modifier>private</modifier>
  <modifier>private</modifier>
  <modifier>private</modifier>
  <modifier>private</modifier>
  <constructor instrumented="false" name="Spreadsheet">
    <modifier>public</modifier>
  </constructor>
  <method instrumented="false" name="setFunction" return-
type="void">
    <modifier>public</modifier>
    <parameter-type>String</parameter-type>
    <parameter-type>Function</parameter-type>
  </method>
  <method instrumented="false" name="getFunction" return-
type="Function">
    <modifier>public</modifier>
    <parameter-type>String</parameter-type>
  </method>
  <method instrumented="false" name="clearFunction" return-
type="void">
    <modifier>public</modifier>
  </method>
  <method instrumented="false" name="installFunctions" return-
type="void">
    <modifier>public</modifier>
  </method>
  <method instrumented="false" name="extractCells" return-
type="Vector">
    <modifier>private</modifier>
    <parameter-type>String</parameter-type>
  </method>
  <method instrumented="false" name="modifyLevels" return-
type="void">
    <modifier>private</modifier>
    <parameter-type>Node</parameter-type>
  </method>
  <method instrumented="false" name="updateLevels" return-
type="void">
    <modifier>private</modifier>
    <parameter-type>Node</parameter-type>
  </method>
  <method instrumented="false" name="setDebug" return-type="void">
    <modifier>public</modifier>
    <parameter-type>boolean</parameter-type>
  </method>
  <method instrumented="false" name="getDebug" return-
type="boolean">
    <modifier>public</modifier>
  </method>
  <method instrumented="false" name="getValue" return-
type="String">
    <modifier>public</modifier>
    <parameter-type>String</parameter-type>
  </method>
  <method instrumented="false" name="setValue" return-type="void">
    <modifier>public</modifier>
    <parameter-type>String</parameter-type>
    <parameter-type>String</parameter-type>
  </method>
  <method instrumented="false" name="setValue" return-type="void">
    <modifier>public</modifier>
  </method>
  <method instrumented="false" name="getNumericValue" return-
type="float">
    <modifier>public</modifier>
    <parameter-type>String</parameter-type>
  </method>
  <method instrumented="false" name="getNode" return-type="Node">
    <modifier>private</modifier>
    <parameter-type>String</parameter-type>
  </method>
  <method instrumented="true" name="evaluateConstant" return-
type="float">
    <modifier>private</modifier>
    <parameter-type>String</parameter-type>
  </method>
  <method instrumented="true" name="evaluate" return-type="void">
    <modifier>private</modifier>
    <parameter-type>Node</parameter-type>
  </method>
  <method instrumented="false" name="calculateFunction" return-
type="float">
    <modifier>public</modifier>
    <parameter-type>Expression</parameter-type>
    <exception-type>FunctionException</exception-type>
  </method>
  <method instrumented="false" name="calculateFunction" return-
type="float">
    <modifier>private</modifier>
    <parameter-type>String</parameter-type>
    <parameter-type>Enumeration</parameter-type>
    <exception-type>FunctionException</exception-type>
  </method>
  <method instrumented="false" name="recalculateSpreadsheet"
return-type="void">
    <modifier>private</modifier>
    <parameter-type>Node</parameter-type>
  </method>
  <method instrumented="false" name="markForRecalculation" return-
type="void">
    <modifier>private</modifier>
    <parameter-type>Node</parameter-type>
  </method>
  <method instrumented="false" name="generateRefreshEvents" return-
type="void">
    <modifier>private</modifier>
  </method>
  <method instrumented="false" name="handleSpreadsheetEvent"
return-type="void">
    <modifier>public</modifier>
    <parameter-type>SpreadsheetEventObject</parameter-type>
  </method>
  <method instrumented="false" name="addSpreadsheetListener"
return-type="void">
    <modifier>public</modifier>
    <modifier>synchronized</modifier>
    <parameter-type>SpreadsheetListener</parameter-type>
  </method>
  <method instrumented="false" name="removeSpreadsheetListener"
return-type="void">
    <modifier>public</modifier>
    <modifier>synchronized</modifier>
    <parameter-type>SpreadsheetListener</parameter-type>
  </method>
  <method instrumented="false" name="fireAction" return-
type="void">
    <modifier>private</modifier>
    <parameter-type>SpreadsheetEventObject</parameter-type>
  </method>
  <method instrumented="false" name="filterCells" return-
type="void">
    <modifier>private</modifier>
    <parameter-type>Vector</parameter-type>
  </method>
</class>
```

APPENDIX B

Output of **CSLprofiler** tool for Spreadsheet component:

```
component Spreadsheet{

public {
   void setFunction (String,Function);
   Function getFunction (String);
   void clearFunction ();
   void installFunctions ();
   void setDebug (boolean);
   boolean getDebug ();
   String getValue (String);
   void setValue (String,String);
   void setValue ();
   float getNumericValue (String);
   float calculateFunction (Expression,String,Enumeration,Node);
   void handleSpreadsheetEvent (SpreadsheetEventObject);
   void addSpreadsheetListener (SpreadsheetListener);
   void removeSpreadsheetListener (SpreadsheetListener);
};

private {
   adaptable float evaluateConstant (String);
   adaptable void evaluate (Node);
};

protected {
};

}
```

Note: *The above output is with respect to the previous output (appendix A) of the AIDE tool given as input given to CSLprofiler tool.*

APPENDIX C

CSL Grammar

Application CSL grammar

Application CSL Starts with a keyword "*application*" and the first line is specified as

Grammar

```
application <application name> {
        Interfaces …
        Properties…
        Components…
        Public/Private/Protected  methods…
}
```

Example

```
application FinalApp {
```

Indicates

Indicates that the application named "FinalApp" is being described in form of CSL.

Note:

- **BOLD** implies compulsory/keyword.
- "< >" implies to be filled by CSL writer.
- [* *] implies optional.
- adapt.spreadsheet.Node like dot " . " separated names imply fully quantified names.
- " ; " is used as a common statement terminator
- " // " is used as a comment statement character

If an application implements some interfaces they are specified in the "implements" clause of the CSL file as follows

Grammar

implements [* <interface1 >, <interface2 >,<interface3 >… *] **;**

Example

implements ActionListener , TableListner ;

Indicates

Indicates briefly the interfaces implemented by the application whose CSL script this is. The word following keyword "implements" is the name of the interface implemented, one can describe more interfaces by giving the name of interfaces separated by a delimiter comma sign " , " as shown in above example.

Implement's clause is typically used to specify the interfaces associated with the application that is described by the current CSL.

Grammar

component <component name> < **instanceof**> **< component type >;**

component <component name> < **extends**> **< component type >;**

component <component name> <**adapts** > **<component / application name > {**

```
        Class
        Actions ….
        Properties …
        Methods …
        Public/Private/Protected  methods …
}
```

Example

```
Component table instanceof TableBean ;
Component spreadsheet extends Spreadsheet;
Component adaptsheet adapts Spreadsheet {
……….
……….
    };
```

Indicates

Indicated if this component described is adaptation or instance. Here the curly braces strictly are used only if the keyword following the <component name > specified by the user is "adapt" else otherwise the normal ";" is to be used.

Note: *Here the rest of grammar included in the Component adaptation is the same as described in the above component CSL grammar description.*

This component area, once description is done is to be terminated by closing curly braces followed by a statement terminator.

Component's are typically used to specify and suggest the required adaptations associated with an external code that is written with the new adaptation functionality indicated by the *class clause* in component. (incase of "adapts" keyword.)

Components are also typically used to specify module associated with a different module that is not in the component domain.(incase of "instanceof","extends" keyword.)

Grammar

property <**Property Type**> <**Property Name**> **;**

Example

property int viewHeight ;

Indicates

Indicates brief description of the data-member of the specified application. In the first line of the CSL description the word following keyword “application” is the name of this application module. Word following keyword property is the data-type of the data member specified and next word following is the name of this property. There are no more attributes to this property.

Properties are accessed through get and set methods. In general, Java strongly encourages class designers to use getProperty() and setProperty() as the name of the methods for accessing Properties. Properties are typically used to specify the variables/data members in the application.

Grammar

public/protected /private {

<**Method Return Type**> <**Method Name**>**(**[* <parameter 1>, <parameter2>,… *] **)**

…………..

};

Example

```
Public {
        void init() ;
        void init_components() ;
        void handleSpreadsheetEvent(SpreadsheetEventObject);
};

Protected {
          void xyz(int a, string node) ;
};

Private {
           void actionresponse (ActionEvent);
           void generateRefreshEvents();
};
```

Indicates

Indicates brief description of the public/protected/private methods specified in the application about which CSL description is, where the area following keyword public/private/protected is the area where the concerned methods declared are described for the given application and they are termed as public/private/protected methods/listeners .the methods and listeners are listed in between the curly braces.

public/protected/private area’s are typically used to specify the methods associated with the concerned declaration type as the keyword suggests respectively ,that is written with respect to the application indicated by the *application clause*.

MORE KEYWORDS

Adaptable

Grammar

adaptable <Method statement > /<Component statement>;

Example

adaptable void generateRereshEvents();

Indicates

Indicates brief description of the method/component specifying whether or not that particular method or component is further adaptable. The first word of a line of the CSL description here has to be “adaptable” implying that the concerned method/module is adaptable.

It makes it easy to know whether the component and/or the method is useful for adaptation in future .

Final

Grammar

final <Method statement > /<Component statement>;

Example

final void generateRereshEvents();

Indicates

Indicates brief description of the method/component specifying whether or not that particular method or component is further adaptable. The first word of a line of the CSL description here has to be “final” implying that the concerned method/module is non-adaptable further.

It makes it easy to know whether the component and/or the method is useful for adaptation in future.

Note: ***Final & Adaptable keywords are mutually contradictory.***

Component CSL grammar

Component CSL Starts with a keyword "*component*" and the first line is specified as

Grammar

```
component  <component name> <adapts > <component / application >
{
        Class
        Actions …
        Properties …
        Methods …
        Public/Private/Protected  methods …
 };
```

Example

Component table instanceof TableBean ;

Component adaptsheet adapts Spreadsheet {

Indicates

Indicated if this component described is adaptation or instance. Here the curly braces strictly are used only if the keyword following the <component name > specified by the user is "adapt" else otherwise the normal ";" is to be used.

Note:

- BOLD implies compulsory/keyword.
- “< >” implies to be filled by CSL writer.
- [* *] implies optional.
- adapt.spreadsheet.Node like dot “ . ” separated names imply fully quantified names.
- “ ; ” is used as a common statement terminator

If a Component adapts some application more details of the component can be specified as follows after opening the curly braces as shown in above example.

This component description further is made of a couple of statements followed by the keywords like class, action, property, method, before, after, negotiate so on.

Grammar

class <**Class Name**> **;**

Example

class Glue ;

Indicates

Indicates the name of the Class where the additional functionality is placed. for example in our case study “Glue” Class.

This class clause is typically used to indicate the external code/class file that is written with the new adaptation functionality associated to this component.

Grammar

property   <Property Type> <Property Name> ;

Example

property int viewHeight ;

Indicates

Indicates brief description of the data-member of the class/module specified component is adapting. In the component clause of the CSL description where the word following keyword “adapts” is the name of this module being adapted.

Note**: *For more details on Property clause see the property section of the CSL application grammar***

Grammar

action <Action Name> ([* <parameter1 type> <parameter 1>, <parameter2 type> <parameter2>,… *] **) ;**

Example

action beforeEvaluate (adapt.spreadsheet.Node n);

action beforeEvaluate (java.lang.Vector refreshList);

Indicates

Indicates brief description of the *method* of the Class/file specified in the "class" clause of the CSL description, where the word following keyword action is the name of this method and is also termed as action name .The attributes of the method are given here in form of parameters in "(" & ")". The parameters are usually a fully quantified name. The parameter generally represent the input datatypes of the method in the given class specified for this component adaptation/description. The parameter type precedes the parameter names.

Actions are typically used to specify the methods associated with the external code that is written with the new adaptation functionality indicated by the *class clause*.

Grammar

method <**Method Return Type**> <**Method Name**>**(**[* <parameter1 type> <parameter 1>, <parameter2 type> <parameter2>,… *] **) {**

```
    negotiate …
    before …
    after …
};
```

Example

```
method void generateRefreshEvents(java.lang.Vector refreshList) {
        negotiate RefreshPolicy:
                filterCells(refreshList);
};

method void evaluate (adapt.spreadsheet.Node node){
      before beforeEvaluate (node);
      after  afterEvaluate (node);
};
```

Indicates

Indicates brief description of the method of the class/module specified component is adapting. In the component clause of the CSL description where the word following keyword “adapts” is the name of this module being adapted.

Word following keyword “method” is the return data-type of the Method specified and next word following is the name of this Method. The attributes of this method are given here in form of parameters in "(" & ")" .the parameters are usually a fully quantified name. The parameter generally represents the input parameters of the method specified. The parameter data-type precedes the parameter name.

Following the method parameter description there is open curly braces where in other concerned description, generally about the adaptations information with respect to this specified method.
This adaptation information is basically based on three keywords, *before, after* and *negotiate*.

BEFORE /AFTER CLAUSE

Grammar

before <before method name > ([* <parameter 1>, <parameter2>,… *] **) ;**

after <after method name > ([* <parameter 1>, <parameter2>,..*] **) ;**

Example

before beforeEvaluate (node);

after afterEvaluate (node);

Indicates

In this clause the action is specified using either of the keyword specifying if the action is to be invoked before the specified method or after the specified method. For example in above example for evaluate method , We specify such that the script suggest to invoke beforeEvaluate action before the method evaluate and on similar lines the script suggest to invoke afterEvaluate action after the method evaluate.

Where in the actions are appropriated coded in the class/code specified in the "class" clause of the CSL description. The actions here are presumed to have the required functionalities to carry out the required adaptation. before & after keywords are typically used to specify and suggest how the adaptations that are required are done using action (atleast specifying to invoke that action) before or after the invocation of the specified method.

NEGOTIATE CLAUSE

Grammar

negotiate <negotiate policy name > :
< action name>([* <parametertype 1>, <parametertype2>,… *] **) ;**

Example

```
negotiate RefreshPolicy :
 filterCells (refreshList);
```

Indicates

In this clause the action is specified with the parameter type associate with the Method specified e.g. *refreshList* in the second line .A unique negotiate name is assigned which follows the keyword “negotiate” which is also termed as negotiate policy name.

Using this negotiate clause one can suggest adaptations such that every time the specified method is invoked the suggested adaptation action filtercells takes the concerned parameter input and process/update it for some required functionality.
Here too the action is supposed to be appropriated coded in the class/code specified in the “class” clause of the CSL description. The actions here are presumed to have the required functionalities to carry out the required adaptation.

The negotiate keyword is typically used to specify and suggest how the adaptations that are required are done using action using the input parameter of the specified method.

Method, once description is done are supposed to be terminated by closing curly braces followed by a statement terminator.

Method clause is typically used to specify how the adaptations that are required are done using the keywords like *before, after* and *negotiate* using some other known information from the class specified in the “class” clause of CSL script.

Note: *Methods are essential and an important part of the component CSL from adaptation point of view one should be thus very careful while writing this portion of CSL since a slight mistake here may result in a logical error which can be hard to debug.*

Grammar

public/protected /private {
<**Method Return Type**> <**Method Name**>**(**[* <parameter 1>, <parameter2>,… *] **)**
……….. **};**

Example

```
Public {
        void init( ) ;
        void init_components( String sp) ;
};
Protected {
        void set_init( ) ;
        void xyz(int a,node refresh) ;
};
```

```
Private {
        adaptable void init_components( String sudeep) ;
         final void xyz(int a,String c) ;
};
```

Indicates

Indicates brief description of the public/protected/private methods specified in the component about which CSL description is, where the area following keyword public/private/protected is the area where the concerned methods declared are described for the given component and they are termed as public/private/protected methods/listeners .the methods and listeners are listed in between the curly braces.

Note: *For more details on Public/Private/Protected clause see the Public/Private/Protected section of the CSL application grammar*

APPENDIX D

CMI RUN OUTPUT AFTER ADAPTATION

- Wednesday March12 2003,
- Adaptation done for adding 10 to each input to calculator

```
Reading Scripts from adaptCMI\scripts
Connect Successful
Storage Directory: c:\program files\ibm\visualage for
java\ide\project_resources\AdaptCMI\adaptCMI\storage
Trying to load: edu.wpi.cs.cs509.calculator.Calculator
ADAPT::::Calculator is adaptable.
Trying to load: edu.wpi.cs.cs509.tutorial.FirstTry
Trying to load: edu.wpi.cs.cs509.calculator.Calculator
ADAPT::::Modified is adaptable.
-----------
public method name       : activate
public method return type: boolean
--  method parameter IBlock
--***  method parameter String
-----------
public method name       : connect
public method return type: boolean
-----------
public method name       : deactivate
public method return type: void
-----------
public method name       : getProvided
public method return type: Enumeration
-----------
public method name       : getRequired
public method return type: Enumeration
--  method parameter int
--  method parameter a
--***  method parameter int
--  method parameter b
-----------
public method name       : multiply
public method return type: int
--  method parameter int
--  method parameter a
--***  method parameter int
--  method parameter b
-----------
public method name       : add
public method return type: int
---  public area over ---
---  public area over ---
---  public area over ---
add component: Calculator
```

```
errorjava.lang.NullPointerException
component Calculator{
// Public area  of application/component
Public {
// Public methods()
   boolean activate( ) ;
   void deactivate( ) ;
   Enumeration getProvided( ) ;
   Enumeration getRequired( ) ;
   boolean connect( IBlock, String ) ;
   int multiply( int, a, int, b ) ;
   int add( int, a, int, b ) ;
};// end of public area
// Private area  of application/component
Private {
// Private methods()
};// end of private area
// Protected area  of application/component
Protected {
// Protected methods()
};// end of protected area
};
//end of componentCalculator
-----------
public method name       : activate
public method return type: boolean
--  method parameter IBlock
--***  method parameter String
-----------
public method name       : connect
public method return type: boolean
-----------
public method name       : deactivate
public method return type: void
-----------
public method name       : getProvided
public method return type: Enumeration
-----------
public method name       : getRequired
public method return type: Enumeration
---  public area over ---
---  public area over ---
---  public area over ---
add component: FirstTry
errorjava.lang.NullPointerException
component FirstTry{
// Public area  of application/component
Public {
// Public methods()
   boolean activate( ) ;
   void deactivate( ) ;
   Enumeration getProvided( ) ;
   Enumeration getRequired( ) ;
   boolean connect( IBlock, String ) ;
};// end of public area
// Private area  of application/component
Private {
// Private methods()
};// end of private area
// Protected area  of application/component
Protected {
// Protected methods()
};// end of protected area
};
//end of componentFirstTry
-----------
action name                : beforeMultiply
action parameter data type: int
action parameter           : a
action parameter data type*: int
action parameter          *: b
 for adapt Method parameter aof type int
 for adapt Method parameter* bof type int
--  method parameter int
--  method parameter a
--***  method parameter int
--  method parameter b
BEFORE method :beforeMultiply
in adapt  method name:multiply
in adapt method return type:int
add component: Modified
which adapts Calculator
>>> Inner parsing: Calculator
-----------
public method name       : activate
public method return type: boolean
--  method parameter IBlock
--***  method parameter String
-----------
public method name       : connect
public method return type: boolean
-----------
public method name       : deactivate
public method return type: void
-----------
public method name       : getProvided
public method return type: Enumeration
-----------
public method name       : getRequired
public method return type: Enumeration
--  method parameter int
--  method parameter a
--***  method parameter int
--  method parameter b
-----------
public method name       : multiply
public method return type: int
--  method parameter int
--  method parameter a
--***  method parameter int
--  method parameter b
-----------
```

```
public method name       : add
public method return type: int
---  public area over ---
---  public area over ---
---  public area over ---
add component: Calculator
errorjava.lang.NullPointerException
continuing...
errorjava.lang.NullPointerException
component Modified adapts Calculator {
   class glue.GlueCalc ;
   action beforeMultiply (int a,int b) ;
   method int multiply(int a,int b){
       before beforeMultiply(int,a,int,b);
    };
};
// end of adaptable component
field name :: Calculator
E : set adapter done
1: class glue.GlueCalc
FirstTry: You have three choices. Either enter three numbers,
separated by a space and press return. If the first number is a 1
I will multiply the two numbers together. If the first number is
a 2 then I will subtract the two numbers. If the first number is
a 3 then I will divide the two numbers.
enter "-1 -1 -1" to stop.
1 2 2
XXX BEFORE invoke call back multiply in Calculator
2 * 2 = 144
1 0 0
XXX BEFORE invoke call back multiply in Calculator
0 * 0 = 100
```

Note:***Above output is after adaptation so it gives***

0*0 =10*10=100 (implies adapted)

APPENDIX E

COMPONENT SPECIFICATION LANGUAGE SCRIPTS

<u>For Spreadsheet Application Adaptation</u>

(Included scripts are just the important ones)

For FinalApp

```
application FinalApp {

implements      ActionListener ,      TableListener ,
                SpreadSheetListener , TextBeanListener ,
                Serializable ;

component vs instanceof ScrollbarBean;
component hs instanceof ScrollbarBean;
component newcomponent instanceof adaptme;
component tb    instanceof TableBean;
component tbBox instanceof TableBean;
component newtb instanceof adapttb;
component app   instanceof FinalApp;
component adaptfa adapts app{
    class Glue;
    action forSheetChange();
    method void doSheetButton(){
     before  forSheetChange();
    };
};//end of component

property int viewHeight;
property int viewWidth;
property CellRegion tableSelected;
property CellRegion tableRSelected;

public {

// public methods
void init();
void init_gui();
void init_components();
// various Listener interface
void actionPerformed(ActionEvent);
void handleTableEvent(TableEventObject);
void handleSpreadsheetEvent( SpreadsheetEventObject);
};
}//end of csl
```

For adaptme component

```
component adaptme adapts spreadsheet{

    class Glue;

    action beforeEvaluate (adapt.spreadsheet.Node node );
    action afterEvaluate (adapt.spreadsheet.Node node );
    action beforeEvaluateSheet(java.lang.String s);
    action beforeEvaluateC (java.lang.String s );
    action beforeEvaluateMail (java.lang.String s );
    action filterCells(java.lang.Vector refreshList);
    action beforeInsert(java.lang.String dest,java.lang.String
                                                         value);

   method void generateRefreshEvents(java.lang.Vector
refreshList) {

         negotiate RefreshPolicy:
         filterCells(refreshList);
   };

   method void evaluate (adapt.spreadsheet.Node node){
         before beforeEvaluate (node);
         after  afterEvaluate (node);
    };

   method void setValue (java.lang.String dest,java.lang.String

value){
         before beforeInsert(dest,value);
    };

   method float evaluateConstant (java.lang.String s){
         before beforeEvaluateSheet(s);
         before beforeEvaluateC (s);
         before beforeEvaluateMail(s);
   };

   };//end of component
```

For Spreadsheet

```
component Spreadsheet{
public {
   void setFunction (String,Function);
   Function getFunction (String);
   void clearFunction ();
   void installFunctions ();
   void setDebug (boolean);
   boolean getDebug ();
   String getValue (String);
   void setValue (String,String);
   void setValue ();
   float getNumericValue (String);
   float calculateFunction (Expression,String,Enumeration,Node);
   void handleSpreadsheetEvent (SpreadsheetEventObject);
   void addSpreadsheetListener (SpreadsheetListener);
   void removeSpreadsheetListener (SpreadsheetListener);
};

private {
   adaptable float evaluateConstant (String);
   adaptable void evaluate (Node);
};

};//end of csl
```

For adapttb

```
component adapttb adapts tb{

    class Glue;

    action afterEvaluateGraphics(java.awt.Graphics g, int
                                         viewCol,int viewRow);

    method void redraw (java.awt.Graphics g, int viewCol, int
                                                    viewRow){
       after  afterEvaluateGraphics( g, viewCol,viewRow);
    };

};//end of component
```

For CMI Adaptation (Included scripts are just the important ones)

For Modified block

```
component Modified adapts edu.wpi.cs.cs509.calculator.Calculator
{

   class glue.GlueCalc;

   action beforeMultiply(int a ,int b);

   // apply the adaptation to the multiply method
   method int multiply (int a ,int b) {
     before beforeMultiply (int a ,int b);
   };
 };
```

For Calculator block

```
component edu.wpi.cs.cs509.calculator.Calculator {
public {
   boolean activate ();
   boolean connect (IBlock,String);
   void deactivate ();
   Enumeration getProvided ();
   Enumeration getRequired ();
   adaptable int multiply (int a,int b);
   adaptable int add (int a,int b);
};

};
```

For FirstTry block

```
component edu.wpi.cs.cs509.tutorial.FirstTry {
public {
   boolean activate ();
   boolean connect (IBlock,String);
   void deactivate ();
   Enumeration getProvided ();
   Enumeration getRequired ();
};

};
```

BIBLIOGRAPHY

[1] George T. Heineman and William T. Councill. *Component-Based Software Engineering: Putting pieces together*. Addison Wesley Longman, Inc., June 2001.

[2] George T. Heineman, “An Experimental Evaluation of Component Adaptation Techniques”, Worcester Polytechnic Institute, November 1999, WPI-CS-TR-99-04.

[3] George T. Heineman, “A Model for Designing Adaptable Software Components”, Pages 121 – 127, Vienna, Austria, August 1998. 22nd Annual International Computer Science and Application Conference.

[4] George T. Heineman, “Adaptation of Software Components”, Technical report, Department of Computer Science, Worcester Polytechnic Institute, April 1999.

[5] George T. Heineman, “Adaptation and Software Architecture”, 3rd Annual International Workshop on Software Architecture (ISAW-3), pages 61-64, Orlando, Florida, FL, USA, November 1998.

[6] George T. Heineman, *Composing Software Systems from Adaptable Software Components*, OMG-DARPA-MCC workshop on Compositional Software Architectures. Monterey, CA, January 1998.

[7] Jan Bosch, “Superimposition: A Component Adaptation Technique”, *Information and Software Technology*, 1999.

[8] Kavita Kanetkar, *Formal Analysis of Component Adaptation Techniques,* M.S.Thesis, Worcester Polytechnic Institute, USA, 2002.

[9] Helgo Ohlenbusch, *Software component adaptation mechanisms*, M.S.Thesis, Worcester Polytechnic Institute, 1999.

[10] Paul Calnan, “AIDE”, SERG, Worcester Polytechnic Institute, 2002.

[11] Grady H. Campbell Jr., “Adaptable Components”, Prosperity Heights Software, www.domain-specific.com, 1999.

[12] George T. Heineman, “Component Model Implementation”, SERG, Worcester Polytechnic Institute, 2002.

[13] M. Woodman, O. Benebiktsson, B. Lefever, F. Stallinger, “Issues of CBD product quality and process quality “, 4th International Workshop of Component-Based Software Engineering at 23rd International Conference on Software Engineering, Toronto, May 2001

[14] Päivi Kallio, Eila Neimelä, “Documented Quality of COTS and OCM Components”, VIT Electronics, Finland.

[15] Chunling Ma, CSL-profiler tool, Independent Study, SERG, Worcester Polytechnic Institute, 2002.

[16] Wojtek Kozaczynski, Composite Nature of Component, Technical Report, Rational Software, USA, 1999.

[17] Ralph Keller and Urs Hölzle, Binary Component Adaptation. Technical Report TRCS97-20, Dept. of computer science, University of California, Santa Barbara, USA, December 1997.

[18] D.M. Yelling and R.E. Strom, Protocol Specifications and Component Adaptors. In ACM Transaction on Programming Languages and Systems, volume 19, pages 292-333, 1997.

[19] George T. Heineman, “Integrating interface Assertion Checkers into Component Models”, Sixth International Component Based Software Engineering workshop (CBSE), Portland, Oregon, USA, May 2003 (To Appear)

[20] Sun Microsystems, Enterprise JavaBeans Specifications, v2.1, *java.sun.com/products/ejb/docs.html*.

[21] N. Wang, D. Schmidt, and C. O’Ryan, “Overview of CORBA Component Model”, in [5], Chapter 31.

[22] Peter Gill, *Probing for Continual Validation Prototype,* M.S.Thesis, Worcester Polytechnic Institute, USA, 2001.

[23] Klaus Bergner, Andreas Rausch, Marc Sihling, Alexander Vilbig, *Adaptation Strategies in Componentware*, ASWEC, Proceedings of t the 2000 Australian Software Engineering Conference, IEEE Computer Society, April 2000.

[24] Noriki Amano and Takuo wantanabe, *An Approach for Constructing Dynamically Adaptable Component-based Software Systems using LEAD++,* OOPSLA’ 99, Workshop on Reflection and Software Engineering, 1999.

[25] Gart T.Leavens, Albert L.Baker and Clyde Ruby, *Preliminary Design of JML: A Behavorial Interface Specification Language for Java*, Iowa State University, USA,1998.

[26] Dae-Kyoo Kim, Sudipto Ghosh, Robert France, Eunjee Song, *Software Component Specification Using Role-Based Modeling Language*, OOPSLA Workshop, USA, 2002.

[27] W. (Voytek) Kozaczynski and Jim Q. Ning, *Concern-Driven Design for a Specification Language Supporting Component-Based Software Engineering*, Proceedings of the 8th International Workshop on Software Specification and Design (IWSSD '96), USA, 1996.

[28] Object Management Group (OMG) IDL, http://www.omg.org/gettingstarted/omg_idl.htm.