ArchAware2:
A Style Based Software Architecture Documentation Tool

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

Saurabh Srivastava

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

June 2011

CERTIFICATE

This is to certify that the work contained in the thesis entitled "*Archaware2: A Style based Software Architecture Documentation Tool*" by *Saurabh Srivastava* has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

June 2011

Prof. T.V. Prabhakar
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
Kanpur 208016

Acknowledgments

I would like to take this opportunity to express my deep sense of gratitude towards **Prof. T.V. Prabhakar**, for providing his able guidance to me. Without his able guidance, it would not have been possible for me to complete my thesis work.

I would also like to thank my friends **Abhinav Mishra**, **Navjot Singh** and **Sumeet Khurana**, for involving themselves in healthy discussions with me, and providing me their valuable inputs from a User's perspective for my work.

Last but not the least, I would like to thank **My parents and family**, for providing me their precious moral support, which was needed by me to complete this thesis work.

Saurabh Srivastava

Department of Computer Science and Engineering,

Indian Institute of Technology, Kanpur

Contents

List of Figures

List of Tables

Dedicated to My Parents

Abstract

Archaware2 is a tool for drawing Software Architecture Diagrams (from here on, whenever we talk about the term "Architecture" or its derived forms like "Architectural", it implies Software Architecture, or related to the same). It extends the already existing Archaware tool, which provides user with the facility of drawing Architectural Views. Archaware2 attempts a different perspective towards expressing Architectures, by providing the user with a handful of well-known Architectural Styles, which can be extended to provide a customized Architecture, as per the need of the user.

Informally, an Architectural Style is a set of Design Decisions, which provide a generic methodology to design the architecture of a system. In general, an Architectural Style is expressed with the help of a Style Template, which describes the elements used in the style, the relationship they share, along with some constraints that should be applied while using the style. Archaware2 provides the user the option to choose a style (current version supports 6 styles) to start the Architecture drawing. A new diagram comes up with a small instance of the Style to start with.

Although the user is free to draw any element part of the style on the diagram, there are certain constraints, dictated by the style template, which must be followed. Because of this, Archaware2 does not allow certain relationships to be drawn, in case; the relationship violates the basic premise of the style. For example, if a certain relationship is fixed to have certain types of components as their source and target, an attempt to draw the relationship involving other components will not be allowed.

Archaware2 also provides Architectural Advices and Warnings to a user, when the user chooses to save a diagram. For example, if there is a loop between the components, the user is warned about the same and given the opportunity to rectify it, in case there's a mistake. It also provides the user with some advice, if the diagram is not properly aligned with the Good Architectural Practices.

Chapter 1

Documenting Software Architecture

In this chapter, we talk about what is Software Architecture, and how do we document it. Section 1.1 introduces the concept of Software Architecture. The next section talks about ways of Documenting Architectures (Section 1.2). We then discuss some of the currently available tools for documenting architectures, including Archaware version 1.0 (Section 1.3). It is followed by a discussion on Archaware2 in brief Section 1.4. Finally, we show how this thesis is organized in Section 1.5.

1.1 Introduction to Software Architecture

There is still no textbook definition of the term *Software Architecture*. The Software Engineering Institute's website[1], provides over 150 defenitions, of practitioners. As defined by Perry and Wolf[2] can be considered a triplet of: *Elements*, *Form* and *Rationale*. Philippe Kruchten[3] added to it, the perspective of non-functional attributes, and defined Software Architecture as:

"It is the result of assembling a certain number of architectural elements in some well-chosen forms to satisfy the major functionality and performance requirements of the system, as well as some other, non-functional requirements such as reliability, scalability, portability, and availability"

The Elements are the actual constituents of the system. This may include Processing Elements, which perform some processing over the data, or the Data Elements, which

hold system data. They can also be Connections, which connect the other kinds of elements. Forms actually represent the options an Architect has, in choosing one element over the other. Forms can be considered as a function which assigns weight values to the elements, which can be used by the architect to categorize elements in categories such as "essential element" or "decorative element". Rationale are the constraints on relationships between the elements. In other words, what combinations of elements are allowed in the architecture, and what are not.

Another refined definition which perceives Architecture from the view of *Principle Design Decisions*, from [4] is:

"Software architecture encompasses the set of significant decisions about the organization of a software system."

This definition assumes that the Architecture of a system can always be expressed as a set of decisions, which are taken by the architect to satisfy the various constraints, keeping in mind, the stakes of the system's stakeholders.

1.2 Documenting the Architectures

Documenting the Architecture is as important as designing it. The architect must document the Architecture in such a way, that it can be referred during downstream processes. Moreover, in case the architect leaves the project, its the Documentation which acts as the only way to understand the Architecture. In general, there are a few important aspects of the Architecture, that an Architect should document:

1. Any Major Design approach, taken to optimize any non functional attribute. For example, if the architect choose to use the Client/Sever approach to make the system more scalable. These approaches can be roughly considered as Architectural Styles or Patterns.

2. The Elements of the System, and their properties. These may be roughly considered as Architectural Views

3. References in Architecture documents, which address the issues related to different Stakeholders of the system.

One of the most commonly used method of describing the elements and their properties, involved in a System, are *Views*. As the name suggests, a view is the snapshot of the system from some angle of perception. In other words, a view generally describes certain parts of a system. Generally, just one view is not enough to visualize the architecture of a system. For example, one view may only show the Physical entities, such as machines, networks, tape devices etc., involved in the system; while some other view may talk about the set of processes and their interactions, constituting the system.

A *Viewset* is generally a set of views, which may be used by an architect, to describe the whole (or at least, a major part) of the system. The Kruchten's 4+1 Viewset[3], is an example.

1.3 Architecture Documentation Tools

There are some dedicated tools available today for design and analysis of Software Architectures.

ArchStudio[5] is an example of the same. It defines an Architecture Description Language of its own, called xADL. The users are provided methods to create and edit Architecture Documents in two types of editors, XML-tree based, and Graphical.

IBM Rational Rose[6] is a general purpose graphical editor for UML. UML can also be used to document architectures, however, their is no concept of a View in UML, and the Architect may have to make heavy usage of Stereotyping.

Archaware[1], is a View-based editor. Archaware is a tool for drawing *Architectural Views* for expressing the Architecture of a Software System. The idea is to equip user with views, with the help of which, all the aspects of the system can be described. The *Kruchten's 4 Views* along with the *Decision View* are a common way to express a System's static as well as runtime behavior. Archaware allows user to draw these views, along with some basic *Component and Connector Diagrams* like a *Hybrid View*.

Another Documentation Tool is Acme Studio[7]. Acme Studio takes a Style based approach towards design of Software Architecture. It allows user to create their own families of Styles, with a support for Armani constraint language for expressing constraints applicable to a particular Style. It provides a comprehensive, (and hence, complex) interface to design Architectures.

[1] Whenever we say Archaware, we mean Archaware version 1.0

1.4 About Archaware2

Just like Acme Studio, Archaware2 also aims to be a Style-based editor. Archaware2 aims to be simpler, and hence, does not provide tweaking to a great extent. A simple example is, defining of *Ports* and *Roles*. If the Architecture Design Tool provides options to the user to define these details, it increases the level of customization, but at the same time, increases the complexity of the overall design process (as in some cases, the behaviour of Roles and Ports may already be well defined, and a Connection is enough to express the idea).

The other approach, with which Archaware2 is developed, is to plug-in, *Good Architectural Practices* in the designing of Architectures. There exist a number of Good Architectural Practices in the field of Software Architecture, which although are not explicitly expressed, but are tried and tested ways to achieve certain quality attributes (or to avoid certain pitfalls), more often than not.

A simple example of a Good Architectural Practice is about *Layering*. One of the important purposes of using a layered approach to architecture design is to provide independence to a Higher Layer with respect to Lower layers. If however, the layers are *bridged*, as shown in Figure 1.1, i.e. a higher layer uses the services of a layer which is not immediately below it in the layered structure, then, we actually sacrifice the *modifiability* of the higher layer to a certain extent.

Another example could be from the domain of Object Oriented Design Methodology. Although *Inheritance* was a phenomenal concept when first conceived, it may not be a great idea to use *Multiple Inheritance*. Multiple Inheritance may sometime result in a problem known as *The Diamond Problem* (Figure 1.2). Such a problem may affect the maintainability of the system.

Archaware2 is designed keeping in mind, that a Style may have similar architectural practices associated with it, and it will be useful, if the user is informed about the same, if the practice is not followed in his diagrams[2].

[2] We use the word *diagram* here to represent any box-and-line documentation of Architecture

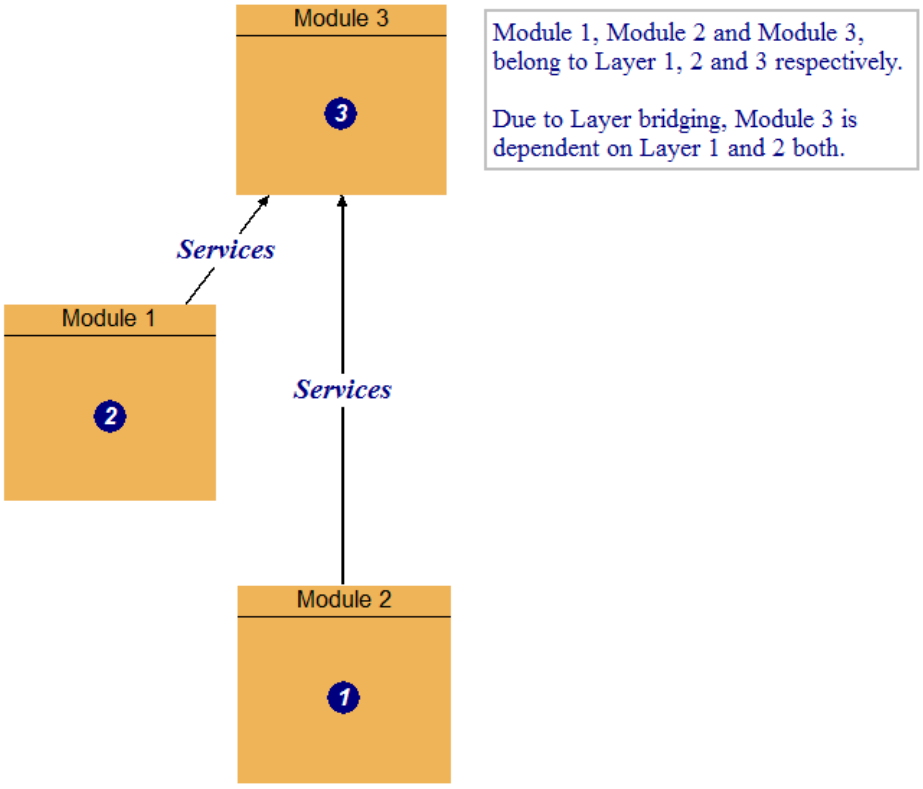

Figure 1.1: A Layered Diagram with bridging

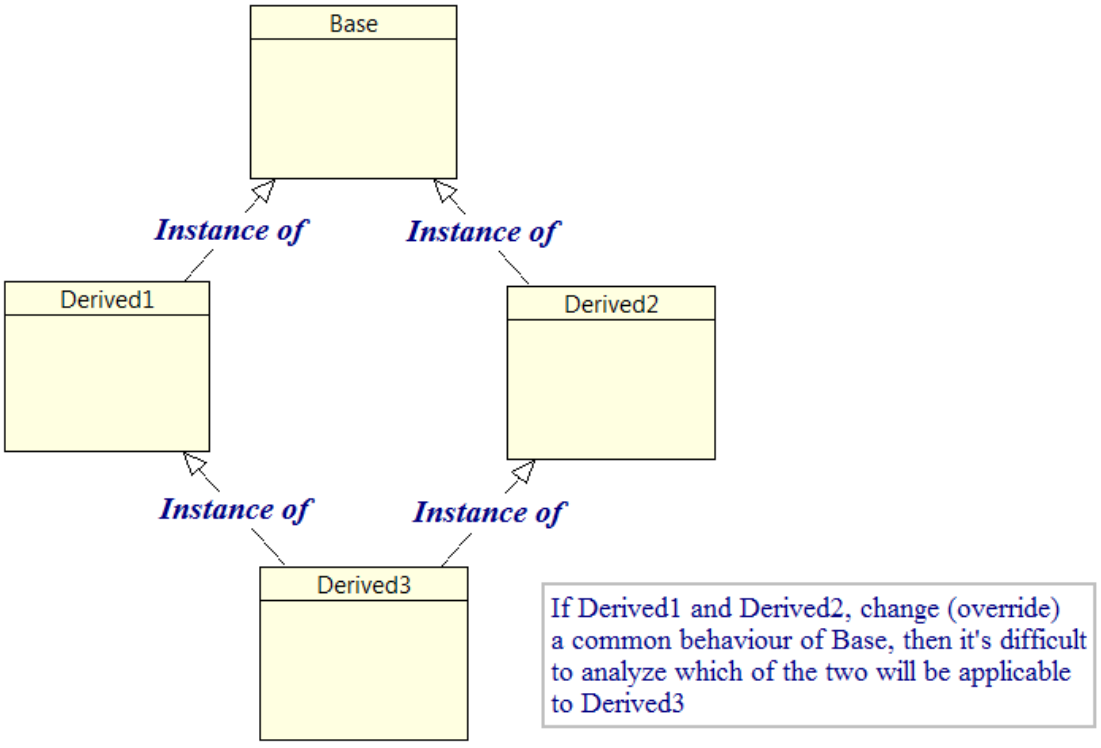

Figure 1.2: Diamond Problem with Multiple Inheritance

1.5 Thesis Outline

The thesis is organized in these chapters:

Chapter 2 describes Archaware2 from a *User's Perspective*. It shows how Archaware2 can be used for documenting Software Architecture, its utility and limitations.

Chapter 3 then describes the technologies used in the development of Archaware2.

Chapter 4 discusses Archaware2 from an *Architect's Perspective*. It talks about the Principle Design Decisions taken while architecting the system. It also explains Module and Component Diagrams of Archaware2.

Chapter 5 and Chapter 6 describes Archaware2 in detail, from a *Developer's Perspective*.

Chapter 7 gives an overview of in-built Examples in Archaware2, meant for guiding user on usage of the system.

Finally, we present the Conclusion and suggest Future Enhancements in the system in Chapter 8.

Chapter 2

Documenting Architectures with Archaware2

In this chapter, we'll discuss in short, the features of Archaware2. We'll discuss the scope of our work, the possibilities and the limitations. We will start by discussing the major functionalities of the System in Section 2.1, and conclude with a discussion on its utility and limitations, as well as the intended audience (Section 2.2).

2.1 Archaware2 Features

In this section, we'll discuss how to create Architectural Diagrams in Archaware2. In general, every Architecture needs at least two different diagrams to document it. A *Module Diagram* describes the basic implementation units, which will collaborate together, to form the System. A *Component Diagram* depicts the major runtime entities, like processes or threads, which will coordinate with each other, to provide the required functionalities, and quality attributes. It may also be required to use an *Allocation Diagram*, which makes sense if the system spans more than one physical machines (otherwise, the Allocation Diagram is said to be *trivial*). Archaware2 currently supports only Module and C&C Styles.

2.1.1 Creating a Module Diagram

As already mentioned, Archaware2 takes a Style based approach towards Architecture Documentations. Module Diagrams can be created using the *Module Styles* provided

in Archaware2. Currently 3 Module Styles are supported (Refer to Chapter 5 for more details). Here are the steps to design a new Moduke Diagram in Archaware2:

a). Click on 'New' in the File Menu.

b). Select the option for Creating a new Module Diagram, under the Archaware2 Diagram Category.

c). Select one of the Styles, and input the File Name and Location, and click Finish.

d). A new Diagram is created, with a sample instance of the Style chosen. The user can import the sample instances of *this Style*, as many times as needed, or create the diagram with the help of the Palette provided on the right side of the Diagram.

e). Once done, the user can save it as a Diagram, export it as image, or convert it to a Custom Style (We'll discuss utility of Custom Styles in a short while).

We'll discuss the Module Styles in detail in Chapter 5. Examples of some Module Diagrams are shown in Figure 5.1, Figure 5.2 and Figure 5.3.

2.1.2 Creating a Component Diagram

Component Diagrams can be created using the *C&C Styles* provided in Archaware2. Currently 3 C&C Styles are supported (Refer to Chapter 5 for more details). Here are the steps to design a new Component Diagram in Archaware2:

a). Click on 'New' in the File Menu.

b). Select the option for Creating a new C&C Diagram, under the Archaware2 Diagram Category.

c). Select one of the Styles, and input the File Name and Location, and click Finish.

d). A new Diagram is created, with a sample instance of the Style chosen. The user can import the sample instances of *this Style*, as many times as needed, or create the diagram with the help of the Palette provided on the right side of the Diagram.

e). Once done, the user can save it as a Diagram, export it as image, or convert it to a Custom Style.

We'll discuss the Component Styles in detail in Chapter 5. Examples of some Component Diagrams are shown in Figure 5.4, Figure 5.5 and Figure 5.6.

2.1.3 Creating a Hybrid Diagram

Sometimes, the Elements and Connections provided in any one of the styles, may not be enough for the User, to document his Architecture. In such a case, the User can choose to create a Hybrid Diagram. A Hybrid Diagram provides access to all the Elements and Connections supported in Archaware2, including both the C&C and Module Styles[1]. There are some extra elements and connections, provided for even greater Customization needs (Discussed in detail in Chapter 5). The steps to create a Hybrid Diagram are:

a). Click on 'New' in the File Menu.

b). Select the option for Creating a new Hybrid Diagram, under the Archaware2 Diagram Category.

c). Input the File Name and Location, and click Finish.

d). A new Hybrid Diagram is created. The User can import any number of instances of *all available Styles*. He can add elements and connections from the Full Palette on the right.

e). A *Custom Style* can be created from, or *imported* in the Current Diagram. Once done, it can be saved or exported to Image.

An example of how a Hybrid Diagram can be used to document an Architecture that doesn't fit in any of the available styles is shown in Figure 7.1.

2.1.4 Drawing Elements and Connections

The Elements and Connections applicable to a particular Style is shown in the Palette on the right of the Editor.

To create a New Element, Click on the element type in the Palette, take the mouse to the required location, click and drag it. When the mouse is released, the element is created.

To create a Connection between two elements, click on the connection, take the mouse to the Source element, and click. Now take the mouse to the target element and click again, the connection is created[2].

User can then change properties of the elements and connections (like element name or

[1] WARNING: The user must be careful, not to mix up elements of one genre (Modules) with that of other (Components)

[2] Only if the connection is legal between the source and target element

connection cardinality), by clicking/double clicking on them. User can also *reconnect* a connection, by changing either its Source or target. This can be done by clicking on either ends of the Connection, and dragging them to the new Connection Point. The connection is changed now[3].

2.1.5 Creating Custom Styles

If the user feels that there is a recurring set of elements and connections, that occur far to often in his Architecture Diagrams, he may be tempted to reuse them rather than creating them every time (e.g. If the applications designed are mostly Web based, the portion of a Browser communicating with a Web Server may be there almost in every diagram). The user may chose to have any such abstract set, and create a Custom Style out of it. The Steps for creating a Custom Style are:

a). Create any of the Style Diagrams; Module, C&C or Hybrid. An existing Diagram can also be opened.

b). Draw the elements and connections needed to be stores as a Custom Style

c). Click on the "Save as Style" option in the Archaware2 Menu, enter the Style Name and Location, and click Save.

d). The Custom Style is created and can now be loaded in other Diagram.

To load a Custom Style[4]:

a). Create or Open a Hybrid Diagram.

b). Click on the "Load Custom Style" option in the Archaware2 Menu, browse to the Style file to be loaded, and click Open.

c). The Custom Style is loaded in the Current Diagram.

Some examples of Custom Styles are shown in Figure 7.2 to Figure 7.6.

2.1.6 Exporting Diagrams to Images

There is a one-click Image Export in Archaware2 for the User Diagrams. The User needs to click the "Export to Image" icon (or the Menu item for the same in Archaware2 Menu).

[3] Only if the reconnection is legal
[4] NOTE: Custom Styles can only be loaded in Hybrid Diagrams

A dialog is shown to input export file name, file type[5] and the location, and click Save. The image of the current diagram in the editor is exported to the image file.

2.1.7 Interpreting Architectural Advices

Archaware2 warns the user whenever the created diagram, may be having a shortcoming[6]. These are just *advices*, given to the user on the basis of a General Opinion in the field of Software Architecture. The user is not bound to accept them. An example of a sample warning is shown in Figure 3.3. It shows an Advice Pop-up from the Layered Style (Sub-Section 5.2.2). The warning is shown at the time of file save operation. The user may choose to click "Cancel" and make changes in the Diagram, or else, if its required, may click on "Continue Saving".

2.2 Archaware2 Scope

We now discuss the scope of Archaware2, in terms of its capabilities and limitations. We will also discuss, who the system can benefit, and why.

2.2.1 Archaware2 Capabilities and Limitations

We can sum up the capabilities of Archaware2 as:

a). Ability to create Architecture Description Diagrams, with a Style as a Starting Point.

b). Ability to create and load custom styles, as per user needs.

c). Ability to export diagrams to Images.

d). A Paint-brush like interface, with minimal detailed inputs is provided.

On the other hand, there are certain limitations on Archaware2 as well:

a). The User Architecture may be too complex to be mapped in any one available style, or sets of styles

b). The User may wish greater customization capabilities than provided.

c). The User may wish to have other export formats, such as XML.

[5] PNG and JPEG formats supported

[6] The word "shortcoming" here means a fallout *in general*, it may be the case that User Architecture is designed in that way only

2.2.2 Usefulness of Archaware2

As explained above, the system has its own pluses and minuses. Archaware2 may be useful for documenting architectures of *Small to Medium* sized software. The primary users could be *students*, who can use the tool, for documenting architectures of their projects. It can help them map their architecture to a more suitably understood Style, rather than drawing a completely random diagram. It may not be suitable for large and complex systems, because of the limitations mentioned in above section.

Chapter 3

The Infrastructure

We will now discuss some of the technologies, that were instrumental in designing Archaware2. We will discuss about the Eclipse Platform in Section 3.1. The next two sections will discuss the Graphical Editing Framework (Section 3.2) and the Draw2d Toolkit (Section 3.3). We'll wrap it up with the Standard Widget Toolkit in Section 3.4.

3.1 The Eclipse Platform

Eclipse is an open source platform for delivering a wide range of functionality. Eclipse Platform is developed in such a way that it can be extended in different ways, to build new tools and platforms. This is possible by making Eclipse a collection of smaller entities, called *Plug-ins*, which collaborate together to form the overall environment[8]. New functionalities can be added, or created from scratch by taking Eclipse Platform as a base, and developing plug-ins or Rich Client Applications over it.

Archaware2 is essentially an Eclipse Plug-in, which uses functionalities of other plug-ins and frameworks provided by the Eclipse Community. The advantage of using Eclipse Platform over designing Archaware2 as an independent Window Application is obvious; it provides the developer a lot of "*ready to use*" functionalities, which would require significant amount of work to be recreated.

Archaware2 makes heavy usage of the *Graphical Editing Framework* (GEF), and *Draw2d Toolkit*. We discuss them in detail in the next section. GEF provides basic support for building Graphical Editors for plug-ins, whereas Draw2d provides basic drawing support on top of the *Eclipse Standard Widget Toolkit* (SWT).

3.2 The Graphical Editing Framework

GEF provides a framework for designing Graphical Editors. GEF is based on the Model-View-Controller (MVC) Architecture (Figure 3.1). The basic functionality GEF provides is a mapping between a *View* and a *Model*, with the help of a *Controller*[9].

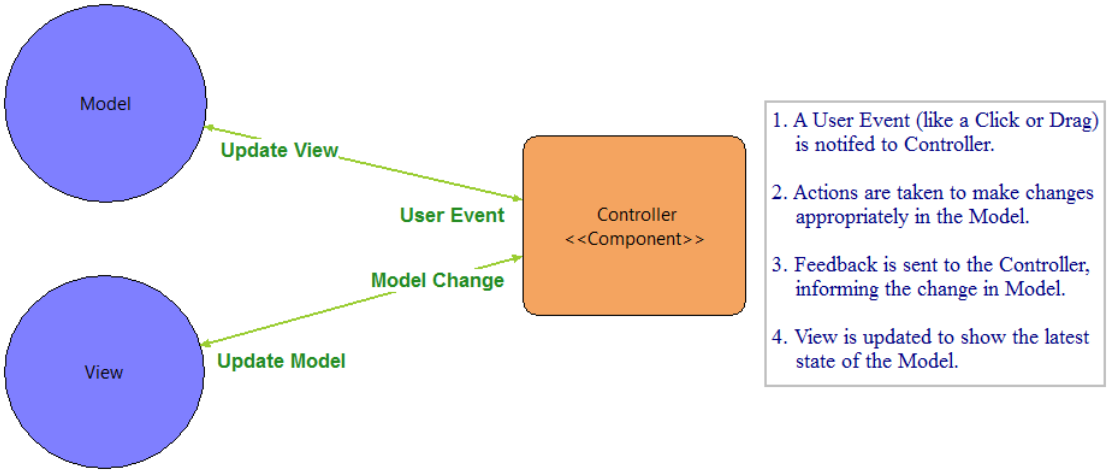

Figure 3.1: Model-View-Controller Architecture

A *Model* is a real world entity, which has properties that differentiates it from other Models in its environment. For Archaware2, all Components, Connectors and Style Diagrams form the entities of our system. So, each one of them have a corresponding Model, which represents the settable properties of these elements. The user, with the help of our Graphical Editor (called the *Archaware2 Editor*) can change these properties.

A *View* is a figure that represents a Model. For instance, we can choose to represent a Server with a Rectangle, an Ellipse or any other complex figure, made up with composition of simpler figures. GEF differentiates between a Model and its associated view, and strongly recommends that the Model should have no dependence on its View. In other words, it should be possible to change the figure for a Model, without making any change in the Model itself.

The *Controller* is the most complex part of the System. It acts like a listener to both User Events as well as a change in some property of the Model. It's the responsibility of the Controller to make the presence of a View transparent to the Model. Generally, there is a one-to-one relationship between the three constituents of the System.

3.3 The Draw2d Toolkit

Draw2d is a toolkit for drawing figures on an SWT Canvas[10]. All the figures are Java Objects and hence, occupy no Operating System resources. Draw2d also captures most of the SWT Events, like click or drag, which can then be caught and handled appropriately.

In the MVC architecture discussed above, the View is provided with the help of Draw2d figures. A figure can be as simple as a Rectangle or as complex as a figure with Labels, Lists, any other Draw2d Widget or even, other figures also. A Draw2d label is a subtype of Draw2d figure, and hence can be seamlessly added to a figure like any other figure, forming a Parent-Child relationship.

In Archaware2, every Component is represented by a specific colored rectangle or ellipse, with other properties such as its name, displayed with the help of Labels (Figure 3.2). Some components, like a *Relation* (Data Element) or a *Layered Module* have other settable properties too, which are represented by Complex figures. For instance, A relation can have *Attributes* (Properties), which are represented by other figure objects. When the User adds an attribute to a Relation, a Parent-Child relationship is created between the Relation and Attribute figure.

The Connectors are designed in a similar fashion, by adding other figures to a base Connection figure. The children of a Connection figure are generally known as Decorations. For instance, in Figure 1.2, the "*instance of*" Connector has a decoration in the middle with a Label, and a hollow triangle arrow decoration at the target end of the Connector.

The Archaware2 Diagrams are actually Draw2d Free Form Layers, which can extend in all directions, and can hold Draw2d Figures as its Children. This is the blank canvas User sees when a New Hybrid Diagram is created (The other diagrams, by default come with some sample figures pre-drawn).

3.4 SWT: The Standard Widget Toolkit

SWT is the toolkit, on top of which Draw2d works. Its a *Widget Toolkit* for Java[11], with the help of which, widgets like Labels and Lists are created. Although the toolkit can be used in a standalone way to build GUI based applications, it is also used quite frequently along with GEF/Draw2d.

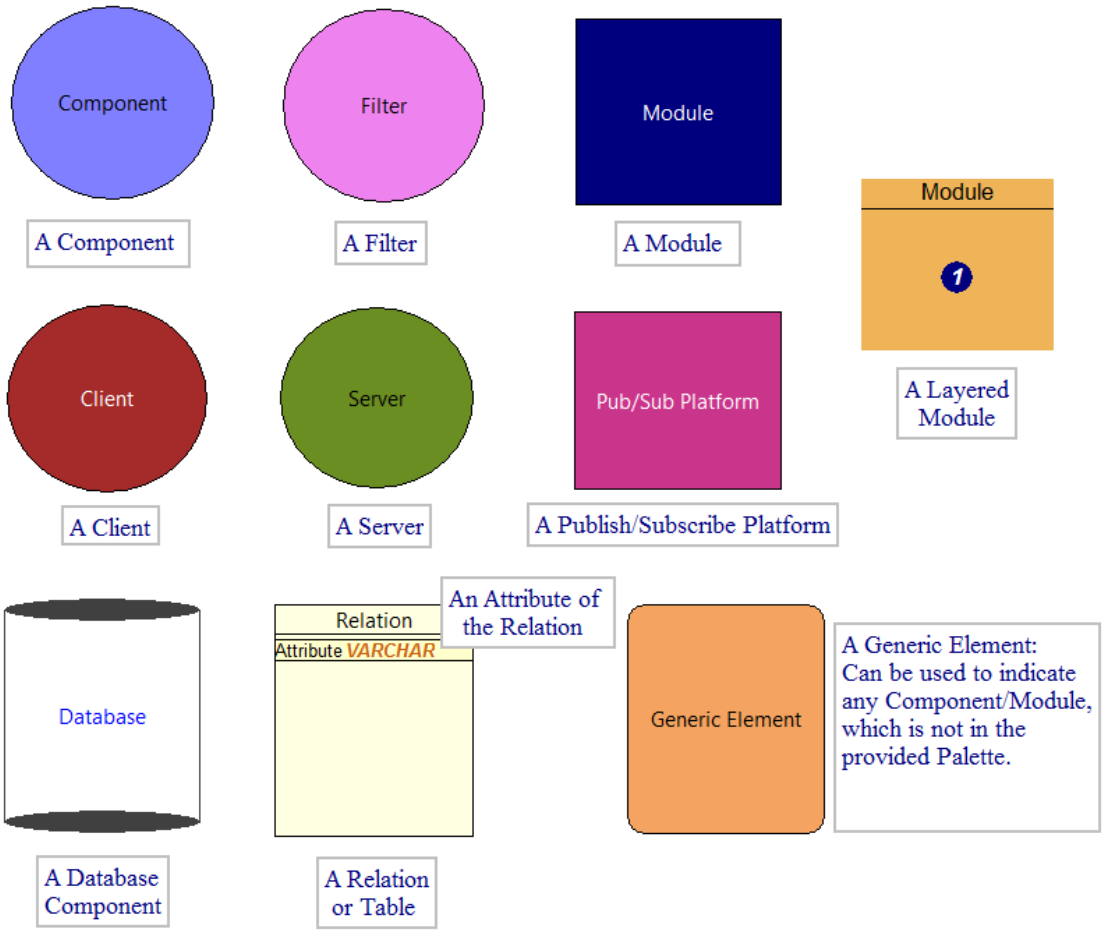

Figure 3.2: The Archaware2 Components

In Archaware2, we make use of the Widget kit mainly to design the *Dialog Boxes* which appear for showing Messages, Errors and Architectural Advices to the User Figure 3.3. We also use the facilities of SWT for designing the *Wizards*, and taking inputs in the form of *Popups*.

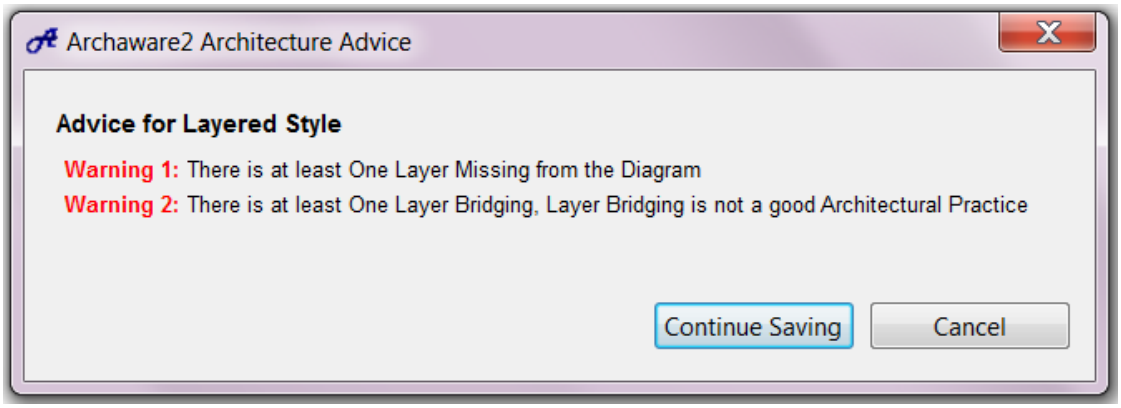

Figure 3.3: Archaware2 Advice for a Layered Diagram

Chapter 4

Archaware2 Architecture

We now describe Archaware2 from an Architectural Perspective. We'll first take a look at the Major Design Decisions which affected Archaware2 architecture (Section 4.1). Next, we will discuss the Major Components of Archaware2 (Section 4.2). Finally, in Section 4.3 we'll discuss how these Components are implemented at the level of Java Packages.

4.1 Principle Design Decisions

When designing an Architecture, there are many factors which may affect the same. For instance, the choice of one technology over the other, may affect the Architecture significantly. Similarly, the concerns of a stakeholder is also be one of the major factors, which affects the Architecture. We discuss a few Design Decisions that we took, along with some reasoning behind the same.

4.1.1 Prefer Extension over Development

Archaware2 is an Eclipse Plug-in. In fact, its an *Eclipse RCP* (Rich Client Platform) Application. An RCP application is very much like any other *Window application*, with the difference that it is actually a stripped down (in most cases) version of the Eclipse IDE. By stripped down we mean, that an RCP is just a *subset* of the Menus, Toolbars, Views, Plug-ins etc. of a regular eclipse IDE.

We could've develop a Window application from the scratch, but it was better to *extend* the flexible Eclipse platform. This is because the application can be further used, or extended, by adding more functionalities, in the form of new Plug-ins (any valid eclipse

plug-in can be a part of Archaware2). Moreover Eclipse provides a basic framework of Windowing, along with a number of built-in functionalities (like Toolbars and Menus), which need not be recreated.

What more we got along by using Eclipse as our base for extension, is the facility to use other existing plug-ins like GEF (Section 3.2) and Draw2d (Section 3.3), which greatly reduced the amount of code required to build Archaware2.

4.1.2 Prefer Simplicity over Completeness

Archaware2 is built with the perspective of providing the user, a near Paint Brush type experience for creating Architectural Documents. What we mean by this, is that the User Interface of the tool should be easy to use, and the User should not be burdened by prompts for the details.

For example, Archaware2 does not provide a "port-to-role" matching logic. Instead, the checks for validness of a connection is put at the creation time, and the connection is created only when the Source and Destination Elements are of allowable types. In other words, ports and roles are implicit entities which are assumed, but never explicitly shown.

This may of course, reduce the completeness of Archaware2 as an Architectural Documentation Tool, but keeping our perspective in account, its a compromise, worth doing.

4.1.3 Use the MVC pattern

This design decision was almost a derivative of the fact that GEF (Section 3.2) supports MVC based architecture. Although you can club the model, view and controller into one or two classes, it is better to follow the convention that GEF advices. This ensures that someone interested in extending Archaware2 and the future developers, find it easy to understand the code base.

Though the pattern GEF advices does not completely fits into the MVC pattern, it can be called a first level derivative of the same. We will discuss about this more in Section 4.2

4.1.4 Starting with the *Shapes Example*

There is a very nice example on learning GEF for beginners, called The Shape Editor Example [12]. Not only does it give you a great overview of GEF/Draw2d, it also can be used as an excellent starting point for building your own Graphical Editor. The example

shows, how to build an editor which can draw a rectangle and an ellipse in a Graphical Editor. It also provides facility to connect any pair of them with either a solid, or a dashed connection.

Since our basic functionality (of drawing shapes and connecting them) was implemented at the basic level in the example, we took the decision to build our application over the Shape Example, extending and tweaking it as and when required.

4.2 Archaware2 Component Description

Figure 4.1 shows a runtime snapshot of Archaware2, for an element, connection, note, or style. As we can see, the architecture has a lot of resemblance to the MVC architecture, with a slight change by inclusion of commands. We briefly describe these Components in this section.

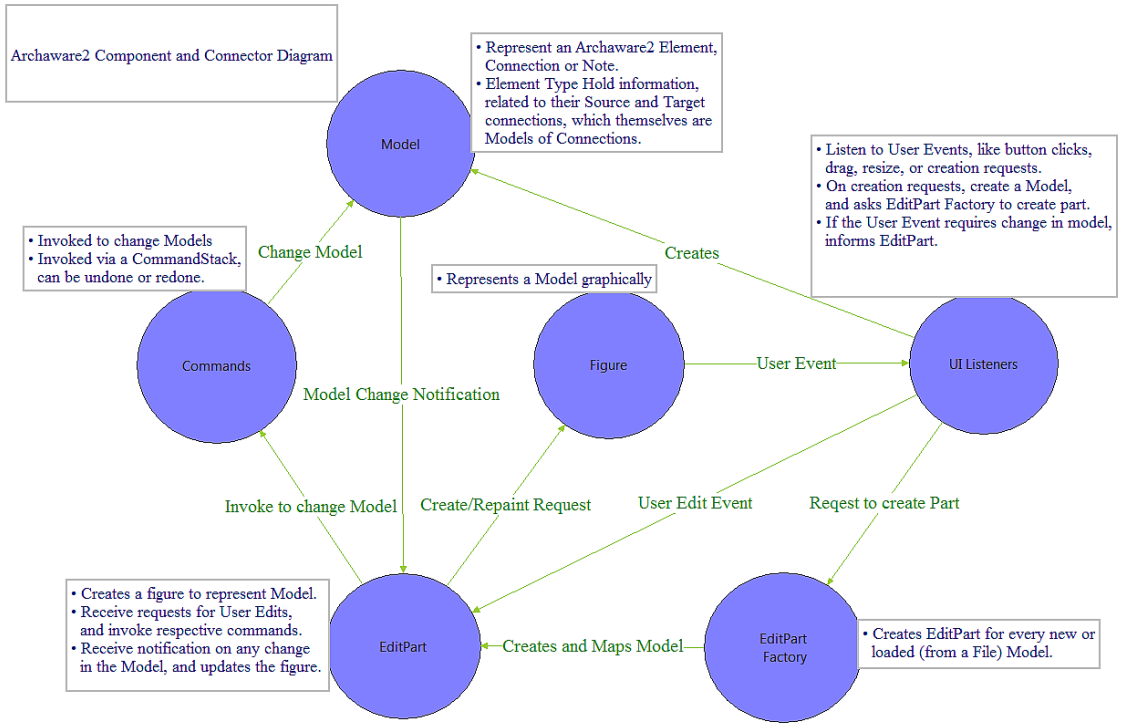

Figure 4.1: Archaware2 Component and Connector Diagram

4.2.1 *Model* Component

A Model is the actual representation of some real world entity. Models have properties, some may be editable, others may not be. When a diagram is stored to a File, *we only save the model.* So, in theory, we can develop different applications, which can depict

the same model in different figures, and the same stored data can be opened in different editors.

For Archaware2, all element models are instances of *Archaware2Element*, which in turn is an instance of Shape. A Shape has properties like width, height and location. An Archaware2Element adds a "name" property to it. The elements then, can add their own properties (like a Layer Number, or set of attributes). Similarly, all Relationships are instances of Connection. A Connection has the property of its style, Solid, or Dashed. The different Connections then add their own properties to it, like annotations, Source or Target Cardinalities, etc. Styles are also represented by their models. If the Architectural Advice, if any, is to be provided with a Style, then its made a part of the Model's behaviour. Notes are special instances of Archaware2Element, which can not have Connections.

4.2.2 *Figure* Component

Every Model is represented by a Figure. The Archaware2 Elements have a Rectangular or Elliptical figure, depending on their type (Component or Module). The Figure is created and managed by the model's Editpart. The model itself is not aware about the figure at all. Figures are direct mode of interactions with the user. They can be selected, moved, resized or deleted by the User. Any such event leads to a change in the Model of that figure, through execution of some command.

4.2.3 *EditPart* Component

The Editpart acts like a mediator between the Model and its Figure. The Editpart act like a listener to both of them. A figure can be changed by the User. When the Editpart gets a change request in this regard, it issues a command, to change the model accordingly. This fires a property change event on the model, on which, the Editpart changes the figure accordingly to match the change in the Model.

4.2.4 *EditPart Factory* Component

The Editpart factory creates editparts on the fly for any provided model. The model could be a freshly created one, or one being loaded from a file. The Editpart factory keeps a Model-to-Part map with it, which is referenced every time, such a request is made to it.

4.3 Archaware2 Module Description

We now take a look at some of the major java packages Archaware2 has, and the functionalities they implement. Figure 4.2 provides a pictorial overview of what we will cover in this section.

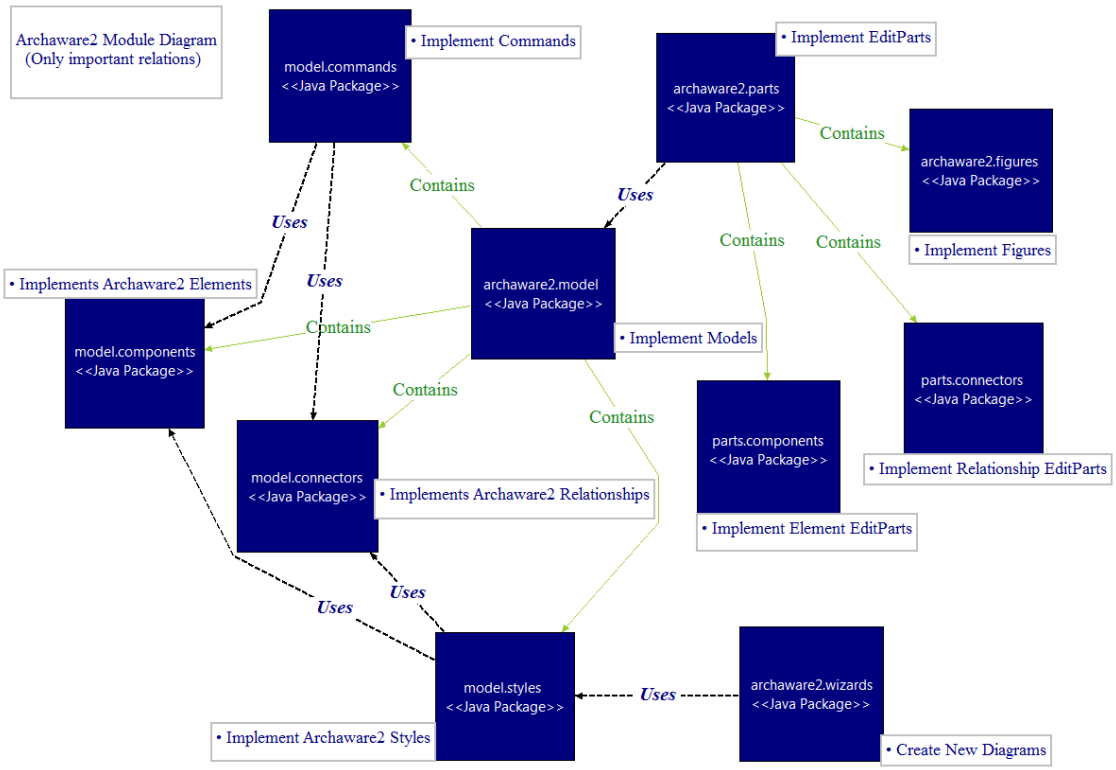

Figure 4.2: Archaware2 Module Diagram (Major Elements and Relationships)

4.3.1 The *model* package

This package contains all the implementations required for the Models that Archaware2 needs. This includes model implementations for the Elements, Connections as well as the Styles. The Package has other packages included in it, which divide the Element implementations, Connection implementations and Style implementations within them. The model package also contains, the package which implements the commands. Although commands can be kept in a different package altogether, but since the only usage of a Command is to change some Model, it can be considered a submodule of the model module in some sense.

4.3.2 The *parts* package

This package contains the implementations for the Controller Components in the MVC pattern. In GEF terminology, a controller is called an Editpart. Generally, a one-to-one mapping between a model and its editpart exists. In Archaware2, we use a same Editpart implementation for all the elements, except the Archaware2 Note, Layered Module, Data Element and Attribute. This is because these elements are slightly different from the other elements (we will discuss the same in Chapter 6). Similarly, we use the same Editpart implementation for all Connections, except Aggregation, m-n, and the two Generic Connections, because of similar reasons.

4.3.3 The *figures* package

This package is responsible for drawing the figures for our models. This includes figures for the elements, as well as the connections. The figures are drawn with the help of a Draw2d class called *Figure*. The Runtime elements, are shown with the help of ellipses. Static elements are shown in rectangles. The Generic Element is shown with the help of a Rounded Rectangle (a generic element can imitate a Module or a Component). Connection figures have decorations attached to them. Decorations are attached to Source, Target and Mid-point locations, depending on the type of connection it is. Connections of different types have different widths and styles (Solid or Dashed).

All elements and connections are assigned unique colors to distinguish them from each other in the Diagram.

4.3.4 The *wizards* and *menuandcommands* packages

The wizards package implements the wizards that are shown to the user for creating a new Archaware2 Diagram. The menuandcommands package is responsible for the working of the Menu items in the Archaware2 Menu, as well as the icons on the Archaware2 Toolbar. These are actually command handlers, which work at the level of Styles, unlike the other commands, which work at the level of elements and connections.

Chapter 5

Archaware2 Styles

In this chapter we take a detailed look at the Styles Archaware2 offers. We start the chapter by introduction to Architectural Styles (Section 5.1). We'll then discuss the two broad category of styles, provided in Archaware2, in Section 5.2 and Section 5.3. Section 5.4 discusses the Hybrid Diagrams, which can be drawn in Archaware2, and Section 5.5 describes the Custom Styles feature. We conclude by discussing the built-in Architectural Advices and Constraints in Section 5.6.

5.1 Introduction to Styles

An *Architectural Style*[1] prescribes a set of Components and Connectors, along with some restrictions on how Connections can be made between them. Styles are generic in nature, in sense; they do not prescribe the applicability of the style to any specific problem (although the Style Templates may give informal advices on its application, without getting too specific to a single case).

Styles can be divided into three Major categories: *Module Styles*, *Component Styles* and *Allocation Styles*[13].

Module Styles define the relationships between *Modules*. A Module is a unit of implementation. A Class that provides a specific functionality may be considered an example of a Module.

Component Styles describe relationship between a set of *Components*. Unlike a Module, a Component is a runtime entity like a Process or a Thread. The Component Styles show the running behavior of a System.

[1] From here on, we use the word *Style* to refer to an Architectural Style

Allocation Styles describe how the Software elements (Modules or Components) are mapped to the overall physical system. In other words, how the Software entities interact with non-software entities like Processors, Communication links, Memory devices etc.

As of now, we have chosen 3 Module Styles and 3 Component Styles to start with in Archaware 2. The Archaware2 architecture is flexible enough to introduce new styles at a later stage, without any significant modification to the current system. The user can start with a sample Style Template and add or remove more elements to it to design the required Architecture.

Archaware2 also provides a *Hybrid Diagram* (Section 5.4) for those users, whose requirements do not map exactly to one of the given styles. The feature of creation and loading of user-defined styles is also provided for users who find a certain style being frequently in their Architecture diagrams.

5.2 Archaware2 Module Styles

As discussed earlier, the major elements of a Module Style are Modules. A Module can be defined as a unit of implementation, which is responsible for providing a set of functionalities. A Module can be anything ranging from a Class or a Structure, to Layers and Tables. A Module Style describes the Static behaviour of a System at Compile time. There are a number of Module Styles known today. Archaware2 currently supports three of them.

5.2.1 The *Uses* Style

The Uses Style is probably the simplest of all styles. The elements of this style are *Modules*, wheres the only Relationship that is used between them is the *Uses* relationship, which shows the dependency of a Module on some other Module. This style is suitable for documenting architectures of Systems being developed in increments. It is also helpful for the Debugging and Testing Phases of the Product Development [13].

An example usage of the style is depicted in Figure 5.1, which describes how a feature of converting text to Lower or Upper Case in a text editor may be implemented.

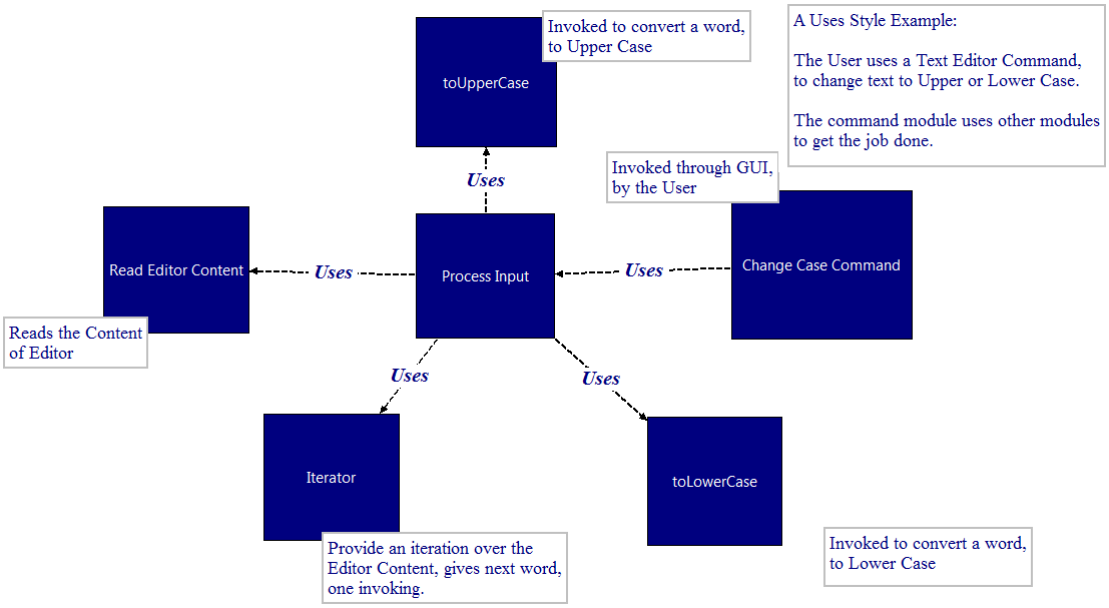

Figure 5.1: An Example of the Uses Style

5.2.2 The *Layered* Style

The Layered Style conceptually divides the system in a set of *Layers*. A Layer can be seen as a group of Modules, working together, to provide a coherent set of Services. Every Layer Services the Layers above it, while taking Services from the Layers below. The types of Relationships in a Layered Style Diagram are; *Services* Relationship, which is always "from a lower Layer to a Higher one", and *Uses* Relationship (same as in Uses Style), between "modules at the Same Layer".

The example of a Layered Style Diagram shown in Figure 5.2, shows a plausible way of architecting a system for Speed Display and Control on an Automobile.

5.2.3 The *Data Model* Style

The purpose of Data Model Style is to model the data that the system will act upon. The basic element of a Data Model Style Diagram is a Data Entity [13]. In Archaware2, we call them *Relations*, because they are very similar to the Tables encountered in Databases. The relations have *Attributes*, which describe their behaviour. The Relationships involved could be *Specialization*, when a Relation is a "Specialization" of other(s); *Aggregation*, when a Relation is a "part of" other relation(s); *1-1* or *m-n* relationships, showing the "cardinality" of the association between two relations; or a *References* relationship between compatible attributes of two relations (similar to a Foreign Key in Databases).

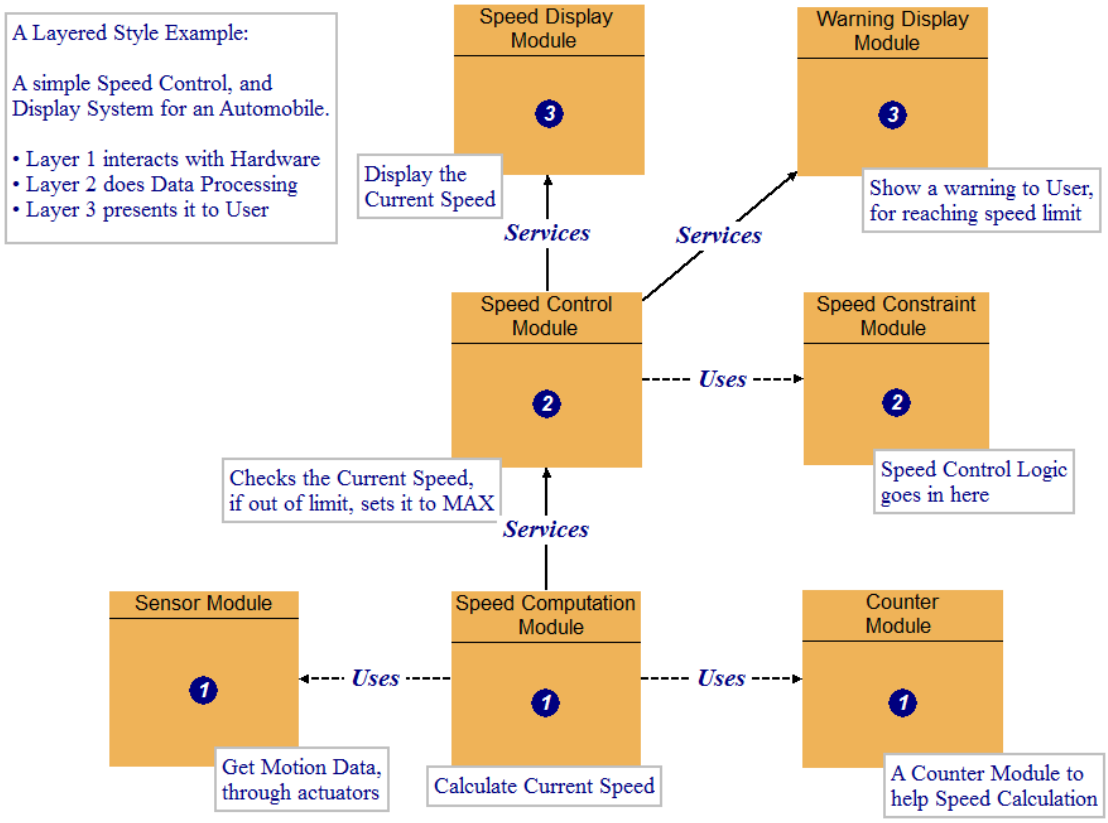

Figure 5.2: An Example of the Layered Style

A simple example from the Banking domain, which can be documented using the Data Model Style, is shown in Figure 5.3.

5.3 Archaware2 Component-and-Connector Styles

The Module Styles depict a system in a static fashion. The Component-and-Connector Styles (also called C&C Styles in short) are meant for capturing the Runtime behaviour of a system. The basic element of a C&C Style is a *Component*. A Component can be any runtime entity like a Process, thread, data store etc.[13] Archaware2 provides support for three C&C Styles currently. We now take a look at the styles currently available in some detail.

5.3.1 The *Client/Server* Style

The Client/Server Style can document scenarios, in which the elements of the system can be divided into sets, i.e. *Clients* and *Servers*. A Client Component is a Component

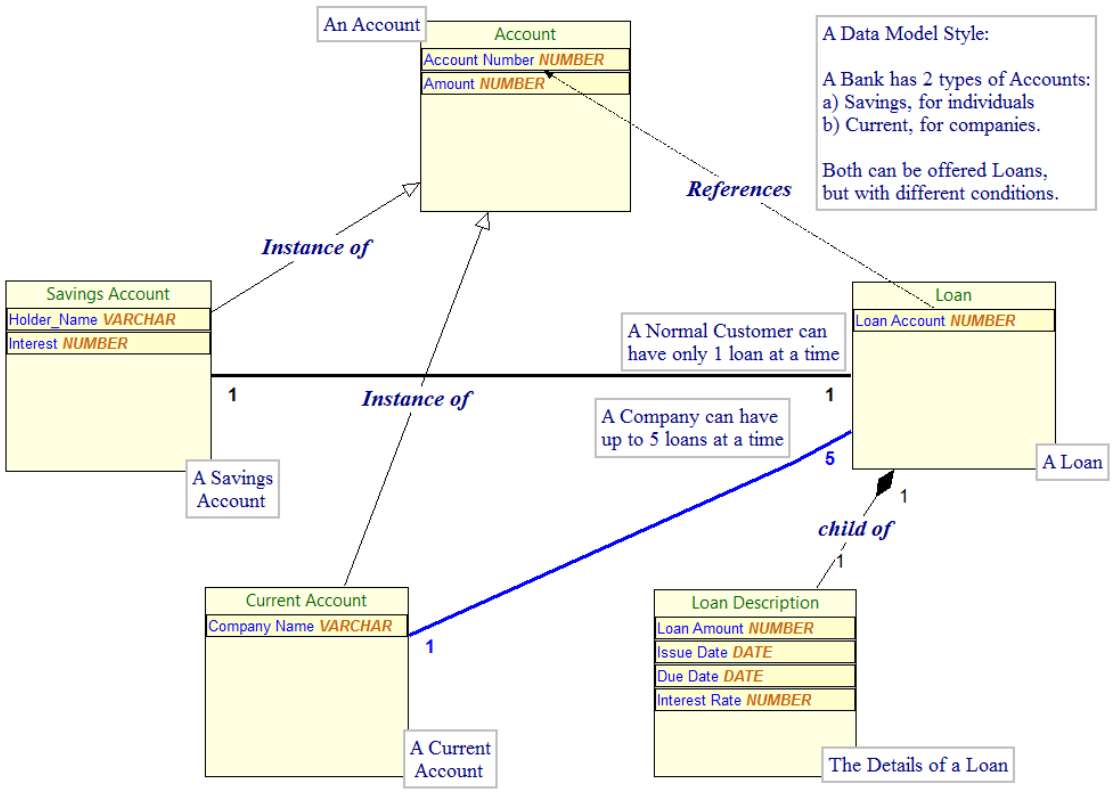

Figure 5.3: An Example of the Data Model Style

which sends Requests to a Server for a Service. A Server Component take Requests from Clients, and send back a reply. Their is only one kind of Relationship defined in a Client Server Style, which is a *Request/Reply* relationship, which connects a Client to a Server. Variations may exist where a Server may act as a Client for some other Server, but Archaware2 doesn't allow the the same (however in the Hybrid Style, the same can be implemented using Generic Elements and Connections).

A simple example of a Multi-threaded Web Server is shown in Figure 5.4.

5.3.2 The *Pipe-and-Filter* Style

The Pipe-and-Filter Style, as the name suggests, is composed of *Pipes* and *Filters*. A Filter is a Component which performs some processing on data, forwarded to it by some other Filter, and passes the processed data to some other Filter. The passing of data between the Filters is done by Pipes. A Pipe is a unidirectional connection between two filters, which provides Data Write facility at one end, and Data Read at the other.

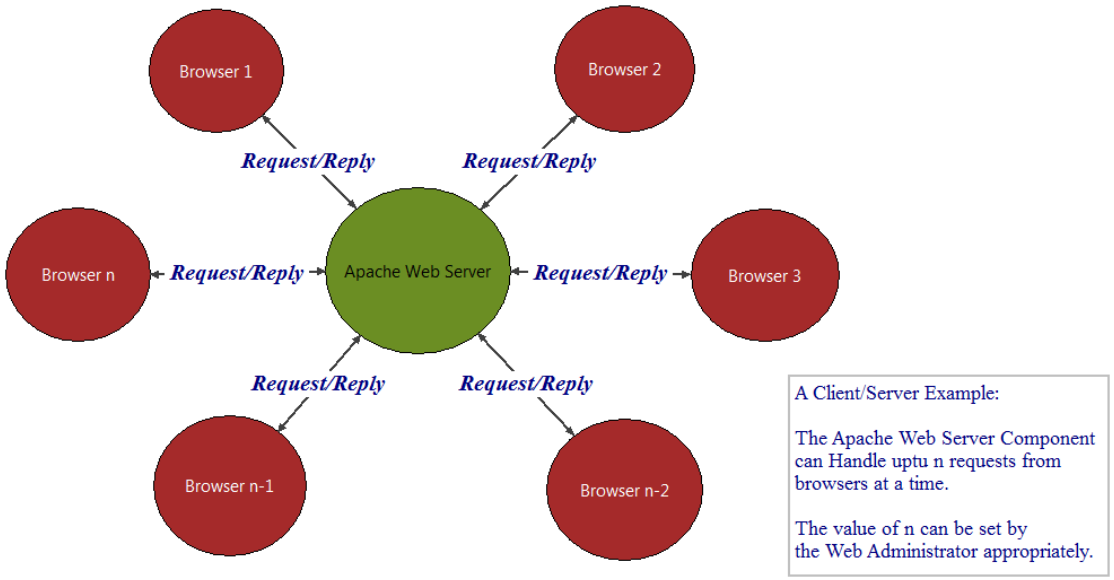

Figure 5.4: An Example of the Client/Server Style

The example shown in Figure 5.5 shows how a set of Pipes and Filters can be deployed to perform updates to a Database.

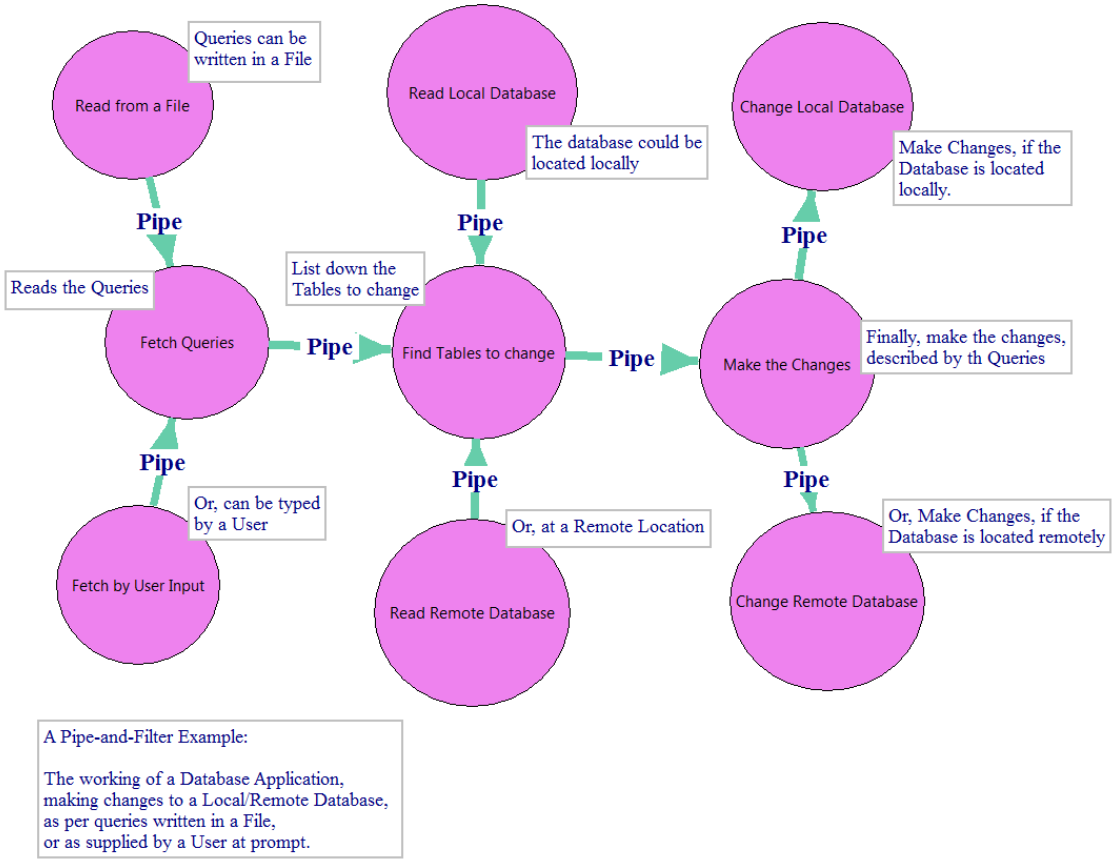

Figure 5.5: An Example of the Pipe-and-Filter Style

5.3.3 The *Publish/Subscribe* Style

The Publish/Subscribe Style is helpful to document the architecture of systems where a set of *Publishers*, publish some data, which is asynchronously accessed by a set of *Subscribers*. A common Platform (In Archaware2 we call it the *Publish/Subscribe Platform*) acts as a mediator between the two types of components. The Relationships involved are a *Publish* relationship, which connects a publisher to the platform; and a *Subscribe* relationship, which connects a subscriber to the platform.

The example of an e-Learning System is shown in Figure 5.6 to give an idea of the Publish/Subscribe Style.

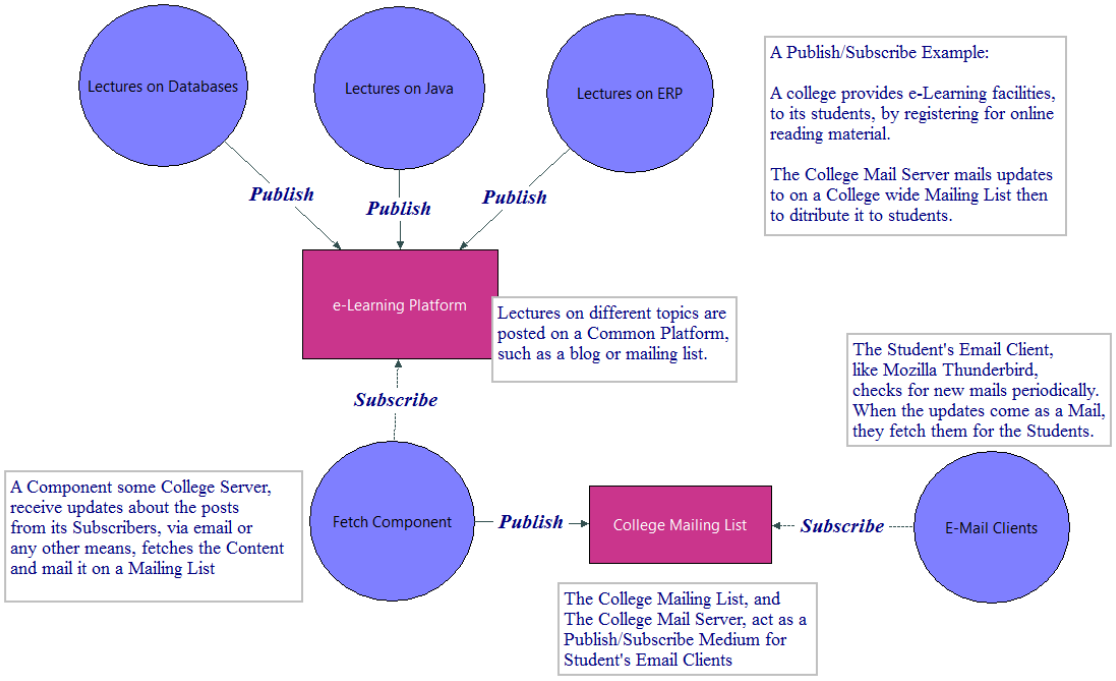

Figure 5.6: An Example of the Publish/Subscribe Style

5.4 Archaware2 Hybrid Diagrams

Although there are a number of Architectural Styles available, it seldom happens that the whole architecture of a System correspond to a single style in its entirety. What may be more natural to think is that the overall architecture may have “instances” of these styles, cooperating with each other to build the overall system. In such cases, any specific style, is not much useful, but a method to combine their instances into a single diagram would rather be more helpful.

The *Hybrid Diagrams* in Archaware2, are probably the kind of diagrams which may suit the user in such situations. A Hybrid diagram can have instances of all the Elements and Connections Archaware2 provides (plus a few more, discussed in a short while). So, a user, for example can have a Client, as well as a Filter in the same diagram. Similarly, A Layered Module and a Publish/Subscribe Platform can also be drawn in the same diagram (though its highly discouraged, as the two depict system behaviour in different ways).

There is however a catch. The Connections can still only be made, in a "valid" fashion, i.e. if you try connecting a Pipe with a Module, it won't. In other words, a Connection can still not be abused. But this restriction can leave us with disjoint sets of style instances, which can never be interconnected, as most of the existing connections can not connect elements from different styles.

To solve this problem, we come up with the idea of a Generic Element and Generic Connectors. A *Generic Element* can be used to imitate any Architectural Element (it is advised that a stereotype, similar to UML, is added in the caption of the element to show its type). Two additional connectors, other than those available in all styles, called *Unidirectional Generic Connection* and *Bidirectional Generic Connection* are also provided in the Hybrid Diagram. Just like a Generic Element can imitate any Element, a Generic Connection can imitate any connection. The User can use any one of the two Connections to connect any two Archaware2 Elements, and give suitable caption(s) to the Connection to express the Connection type. However, the responsibility of using the Generic Elements and Connections and whether they make sense or not, lies totally with the user. In other words, if the user is using the Hybrid diagram, we expect that the user is good enough to understand basics of Software Architecture.

We will look at some examples in **Archaware2 Examples** (Chapter 7) about using a Hybrid Diagram to solve a similar problem.

5.5 Archaware2 Custom Styles

As we discussed in the previous section, it may often be the case that a single style is not good enough for a User. Moreover, a user may be tempted to document a "recurring pattern"[2] within his diagrams, and may wish to have it *stored* at some place, from where, it can be fetched as many times as needed. The Archaware2 Custom Styles are just

[2]Not to be confused with *Architectural Patterns*.

meant for this. *Archaware2 Custom Styles* can be used to save any arbitrary elements and connections to a file, which can be loaded in a Hybrid Diagram later[3]. This essentially means, a User can create an arbitrary style on his own, and use it subsequently in his diagrams.

We will look at some examples of how Custom Styles can be used to document some other popular styles in **Archaware2 Examples** (Chapter 7).

5.6 Archaware2 Advices and Constraints

As mentioned before, every Style has a set of Constraints and Cautions associated with it. Archaware2 implements them at 2 levels. The *Constraints* are implemented in the form of prohibited connections between a pair of Elements. For example, its not possible to connect an module to another module at a lower or same layer with a Services connection, in the Layered Style (Sub-Section 5.2.2). Similarly in the Publish/Subscribe Style (Sub-Section 5.3.3), a subscriber can not make a Subscribe connection to a publisher, it can only do so with a publish/subscribe platform.

Architectural *Advices* are hints to a user, that something in the drawn diagram may not be a good idea. For example, if there is an instance of Multiple Inheritance in the Data Model Style (Sub-Section 5.2.3), the User will be warned of the same (Multiple Inheritance is not considered a Good Practice, as it generally reduces Maintainability of the System). Similarly, if the pipes and filters in a Pipe-and-Filter Style (Sub-Section 5.3.2) has a loop, the user is informed about the same, as such a system may never exist practically.

The difference between a constraint and an advice is, a *Constraint is always maintained*, whereas *acting on an advice is optional* for the User. The advices are presented to the User at the time of Saving the file. The user can cancel the Save and restructure diagram, or can continue saving it without any restructuring. However, there are *no advices* in a Hybrid Diagram (Section 5.4), since the User is free to draw anything, which may not be close to any of the Styles we mentioned.

[3]Since Custom Styles may have any elements and connections, they can only be imported to Hybrid Diagrams

Chapter 6

Archaware2 Elements and Connectors

Archaware2 provides the user a Palette of Elements and Connections, which can be clicked, an drawn on the diagram canvas. Only those elements and connections, which are applicable to a Style, are shown in the Palette, while rest are hidden. In this chapter, we will discuss all the Archaware2 Elements and Connections in brief. Section 6.1 will discuss the Elements, while Section 6.2 will discuss the Connections in Archaware2.

6.1 Archaware2 Elements

All Archaware2 Elements inherit Size and Location properties (Sub-Section 4.2.1). Some Elements have other properties too. We'll discuss them in this section.

6.1.1 Component

A Component is the basic element of C&C Styles. Refer to Figure 6.1, the Ellipses in the diagram are instances of a Component. Table 6.1 summarizes the Component element.

Behaviour	Runtime entity
Properties	Name
Figure	Ellipse
Figure Background	Light Blue
Figure Foreground	Black

Table 6.1: The *Component* Element

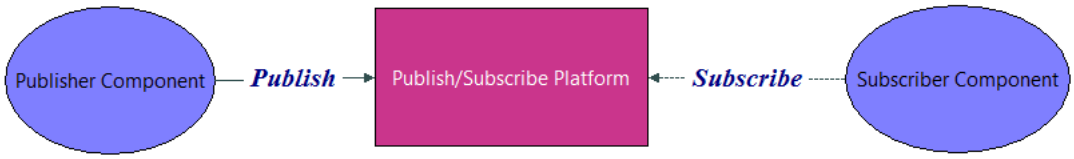

Figure 6.1: Example showing: Components, Pub/Sub Platform, Publish and Subscribe Connections

6.1.2 Publish/Subscribe Platform

A Publish/Subscribe Platform (or Pub/Sub Platform in short) is the platform which enables the Publishing and Subscribing Actions. It may be a network, mailbox, or even a Shared Memory Space. Refer to Figure 6.1, the rectangle in the diagram is an instance of a Pub/Sub Platform. Table 6.2 summarizes the Pub/Sub Platform element.

Behaviour	Storage and/or Communication entity
Properties	Name
Figure	Rectangle
Figure Background	Dark Pink
Figure Foreground	White

Table 6.2: The *Pub/Sub Platform* Element

6.1.3 Client Component

A Client is a special type of Component, used in a Client Server Style. It sends "Requests" to a Server Component (Sub-Section 6.1.4), and gets a "Reply" back. Refer to Figure 6.2, the Reddish Brown ellipse in the diagram is an instance of a Client Component. Table 6.3 summarizes the Client element.

Behaviour	Runtime entity
Properties	Name
Figure	Ellipse
Figure Background	Reddish Brown
Figure Foreground	White

Table 6.3: The *Client* Element

Figure 6.2: Example showing: Client, Server and Request/Reply Connection

6.1.4 Server Component

A Server is a special type of Component, used in a Client Server Style. It receives "Requests" from a Client Component (Sub-Section 6.1.3), and sends a "Reply" back. Refer to Figure 6.2, the Greenish ellipse in the diagram is an instance of a Server Component. Table 6.4 summarizes the Server element.

Behaviour	Runtime entity
Properties	Name
Figure	Ellipse
Figure Background	Olive Drab (Shade of Green)
Figure Foreground	White

Table 6.4: The *Server* Element

6.1.5 Filter Component

A Filter is a special type of Component, used in a Pipe-and-Filter Style. It receives data from other Filters with the help of Pipes, do some processing on it, and then pass it on to the next Filter, via another Pipe. Refer to Figure 6.3, the ellipse in the diagram is an instance of a Filter Component. Table 6.5 summarizes the Filter element.

Behaviour	Runtime entity
Properties	Name
Figure	Ellipse
Figure Background	Light Pink
Figure Foreground	Black

Table 6.5: The *Filter* Element

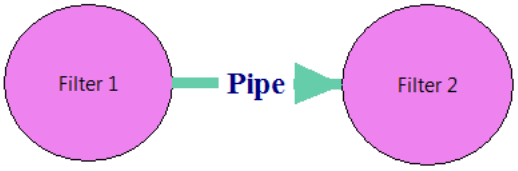

Figure 6.3: Example showing: Filters and Pipe Connection

6.1.6 Module

A Module is the basic element of the Module Styles. It could be a Class, a Package, a C file or any other Implementation unit. Refer to Figure 6.4, the rectangle in the diagram is an instance of a Module. Table 6.5 summarizes the Module element.

Behaviour	Implementation unit
Properties	Name
Figure	Rectangle
Figure Background	Dark Blue
Figure Foreground	White

Table 6.6: The *Module* Element

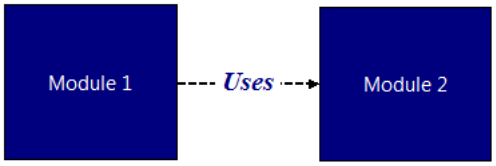

Figure 6.4: Example showing: Modules and Uses Connection

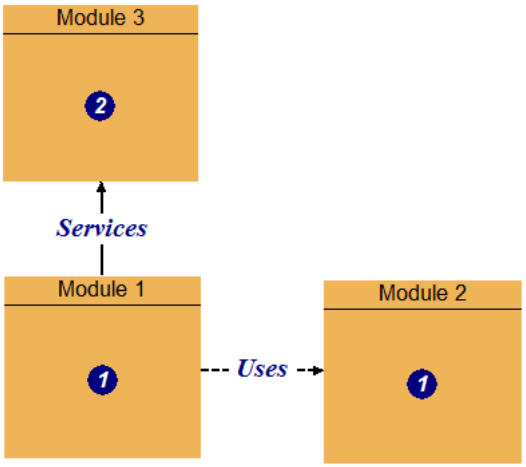

Figure 6.5: Example showing: Layered Modules, Services and Uses Connections

6.1.7 Layered Module

A Layered Module is a special type of Module, used in a Layered Style. Every Layered Module, belongs to a logical layer. Refer to Figure 6.5, the rectangle in the diagram is an instance of a Layered Module. Table 6.7 summarizes the Layered Module element.

Behaviour	Implementation unit
Properties	Name, Layer Number
Figure	Rectangle
Figure Background	Light Orange, Dark Blue for Layer
Figure Foreground	Black, White for Layer

Table 6.7: The *Layered Module* Element

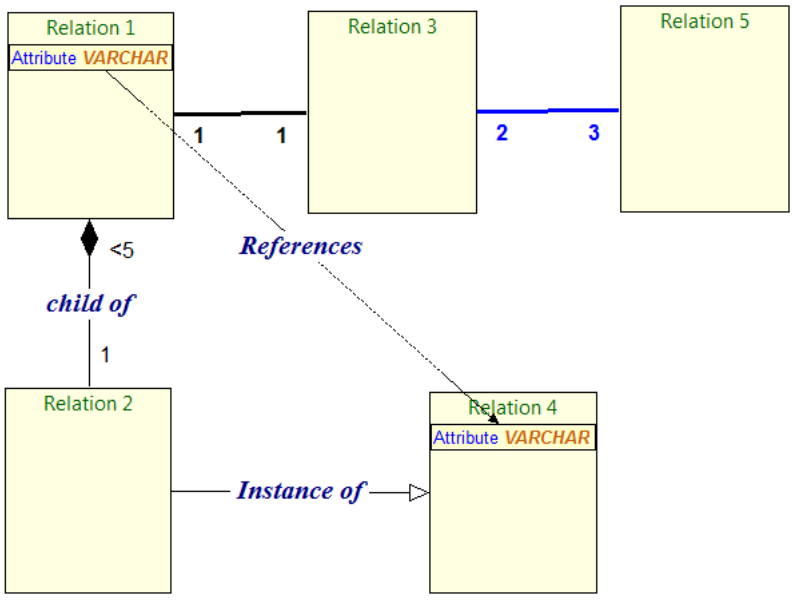

Figure 6.6: Example showing: Relations, Attributes; Specialization, Aggregation, 1-1, m-n and References Connections

6.1.8 Relation

A Relation or Data Element is a special type of Module, used in a Data Model Style. It represents a Data Entity, with Attributes (Sub-Section 6.1.9). It can be specialization or child of other Relations too. Refer to Figure 6.6, the bigger rectangles in the diagram are instances of Relation. Table 6.8 summarizes the Relation element.

Behaviour	Data Modelling
Properties	Name, Attributes
Figure	Rectangle
Figure Background	Light Yellow
Figure Foreground	Green

Table 6.8: The *Relation* Element

6.1.9 Attribute

An attribute is a child entity of a Relation (Sub-Section 6.1.8), used in a Data Model Style. It represents some property of the Relation. Attribute has a propert called Data Type, which can be one of the three values, "NUMBER", "VARCHAR" or "DATE". Refer to Figure 6.6, the smaller rectangles within the relations are instances of Attributes. Table 6.9 summarizes the Attribute element.

Behaviour	A Property of Relations
Properties	Name, Data Type
Figure	Rectangle
Figure Background	Light Yellow
Figure Foreground	Blue, Dark Orange for Data Type

Table 6.9: The *Attribute* Element

6.1.10 Database

A Database is a Data Storage Entity. Its available currently in only Hybrid Diagrams. Refer to Figure 6.7, the figure of a Cylinder is an instance of Database.
Table 6.10 summarizes the Database element.

Behaviour	Storage Element
Properties	Name
Figure	Cylinder
Figure Background	White, Gray Tops
Figure Foreground	Blue

Table 6.10: The *Database* Element

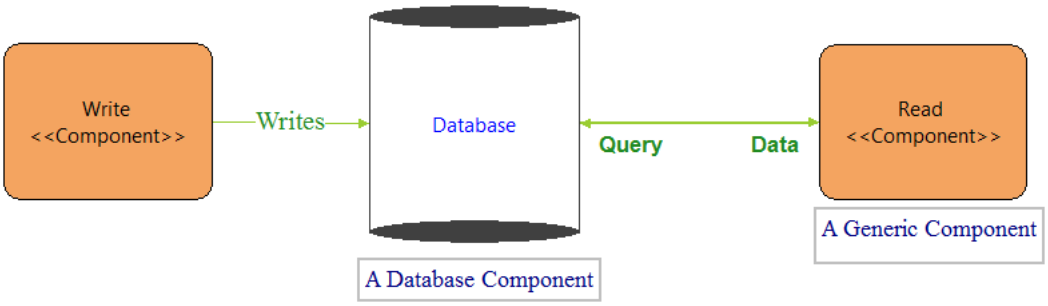

Figure 6.7: Example showing: Database, Generic Component; Generic Connections

6.1.11 Generic Element

A Generic Element is an element which can be molded into anything. The User is free to use it as any element he wants. It is useful in cases when none of the elements suit the User's requirements. They are only available in the Hybrid Diagrams.

Refer to Figure 6.7, the figures of Rounded Rectangles are instances of Generic Elements. Table 6.11 summarizes the Generic Element.

Behaviour	Anything as per requirements
Properties	Name
Figure	Rounded Rectangle
Figure Background	Dark Orange
Figure Foreground	Black

Table 6.11: The *Generic* Element

Archaware2 Note

Although Archaware2 Note is not really an element, its properties are very similar to that of an Archaware2 Element. The only difference is, it is used to write descriptions, and hence, no connections can be made to it. Table 6.12 summarizes the Archaware2 Note.

Behaviour	User Description Tool
Properties	Name
Figure	Rectangle
Figure Background	White, with gray border
Figure Foreground	Blue

Table 6.12: The *Archaware2 Note*

6.2 Archaware2 Connections

All Archaware2 Connections have Source and Target Properties (Sub-Section 4.2.1). There are other properties that they have too. In this section, we'll discuss them

6.2.1 Publish Connection

The Publish Connection is used by a Publishing Component to publish its data on to a Pub/Sub Platform in the Publish/Subscribe Style. It is shown in Figure 6.1. Table 6.13 summarizes the Publish Connection.

Style	Solid
Source Annotation	None
Target Annotation	None
Mid-Point Annotation	Publish
Figure Color	Dark Slate Gray
Annotation Foreground	Dark Blue

Table 6.13: The *Publish* Connection

6.2.2 Subscribe Connection

The Subscribe Connection is used by a Subscribing Component to fetch data from a Pub/Sub Platform in the Publish/Subscribe Style. It is shown in Figure 6.1. Table 6.14 summarizes the Subscribe Connection.

Style	Dashed
Source Annotation	None
Target Annotation	None
Mid-Point Annotation	Subscribe
Figure Color	Dark Slate Gray
Annotation Foreground	Dark Blue

Table 6.14: The *Subscribe* Connection

6.2.3 Request/Reply Connection

The Request Reply Connection is used by a Client to send request and receive response from the Server in the Client/Server Style. It is shown in Figure 6.2. Table 6.15 summarizes the Subscribe Connection.

Style	Solid
Source Annotation	None
Target Annotation	None
Mid-Point Annotation	Subscribe
Figure Color	Dark Gray
Annotation Foreground	Dark Blue

Table 6.15: The *Request/Reply* Connection

6.2.4 Pipe Connection

The Pipe Connection is used by a Filters to send and receive data in the Pipe-and-Filter Style. It is shown in Figure 6.3. Table 6.16 summarizes the Pipe Connection.

Style	Solid
Source Annotation	None
Target Annotation	None
Mid-Point Annotation	Pipe
Figure Color	Medium Aquamarine (A Shade of Pink)
Annotation Foreground	Dark Blue

Table 6.16: The *Pipe* Connection

6.2.5 Uses Connection

The Uses Connection is used by Modules to depict their usage of other modules in the Uses and Layered Styles. It is shown in Figure 6.4. Table 6.17 summarizes the Uses Connection.

Style	Dashed
Source Annotation	None
Target Annotation	None
Mid-Point Annotation	Uses
Figure Color	Black
Annotation Foreground	Dark Blue

Table 6.17: The *Uses* Connection

6.2.6 Services Connection

The Services Connection is used by Layered Modules to depict their usage by Layered Modules at higher Layers in the Layered Style. It is shown in Figure 6.5. Table 6.17 summarizes the Services Connection.

Style	Solid
Source Annotation	None
Target Annotation	None
Mid-Point Annotation	Services
Figure Color	Black
Annotation Foreground	Dark Blue

Table 6.18: The *Services* Connection

6.2.7 Aggregation Connection

The Aggregation Connection is used by Relations to show that another relation is a part of this relation in the Data Model Style. It is shown in Figure 6.6. Table 6.19 summarizes the Aggregation Connection.

Style	Solid
Source Annotation	<Source Cardinality>
Target Annotation	<Target Cardinality>
Mid-Point Annotation	Child of
Figure Color	Black
Annotation Foreground	Dark Blue

Table 6.19: The *Aggregation* Connection

6.2.8 Specialization Connection

The Specialization Connection is used by Relations to show that another relation is a the parent of this relation in the Data Model Style. It is shown in Figure 6.6. Table 6.20 summarizes the Specialization Connection.

Style	Solid
Source Annotation	None
Target Annotation	None
Mid-Point Annotation	Instance of
Figure Color	Black
Annotation Foreground	Dark Blue

Table 6.20: The *Specialization* Connection

6.2.9 References Connection

The References Connection is used by Attributes to show that another attribute defines the set of valid values for it in the Data Model Style. It is shown in Figure 6.6. Table 6.21 summarizes the References Connection.

Style	Dashed
Source Annotation	None
Target Annotation	None
Mid-Point Annotation	References
Figure Color	Black
Annotation Foreground	Dark Blue

Table 6.21: The *References* Connection

6.2.10 1-1 Connection

The 1-1 Connection is used by Relations to show that another relation has a connection cardinality of one-one in the Data Model Style. It is shown in Figure 6.6. Table 6.22 summarizes the 1-1 Connection.

Style	Solid
Source Annotation	1
Target Annotation	1
Mid-Point Annotation	None
Figure Color	Black
Annotation Foreground	Black

Table 6.22: The *1-1* Connection

6.2.11 m-n Connection

The m-n Connection is used by Relations to show that another relation has a connection cardinality of many-many in the Data Model Style. It is shown in Figure 6.6. Table 6.23 summarizes the m-n Connection.

Style	Solid
Source Annotation	<Source Cardinality>
Target Annotation	<Target Cardinality>
Mid-Point Annotation	None
Figure Color	Blue
Annotation Foreground	Blue

Table 6.23: The *m-n* Connection

6.2.12 Generic Connections

The Generic Connections are meant to take place of any connection. It is useful if the Connection required by the User is not available in the palette, or the User wants to join elements present in two different styles. There are two Generic Connections, one Unidirectional, the other one, Bidirectional. They are available only in the Hybrid Diagrams. They are shown in Figure 6.7. Table 6.24 and Table 6.25 summarize the Generic Connections.

Style	Solid
Source Annotation	None
Target Annotation	None
Mid-Point Annotation	<User Defined>
Figure Color	Green
Annotation Foreground	Green

Table 6.24: The *Generic Unidirectional* Connection

Style	Solid
Source Annotation	<User Defined>
Target Annotation	<User Defined>
Mid-Point Annotation	None
Figure Color	Dark Green
Annotation Foreground	Dark Green

Table 6.25: The *Generic Bidirectional* Connection

Chapter 7

Archaware2 Examples

We have already seen examples of the Archaware2 Styles. In this Chapter, we see an example of the Archaware2 Hybrid Diagram (Section 7.1). Then, we take a look at some other styles, which are defined with the help of Hybrid Diagrams, and then converted to Archaware2 Custom Styles (Section 7.2). These Style Diagrams, along with the Example Diagrams we saw in Chapter 5, are built-in examples under Archaware2. The user can import them in his workspace, and can modify them or use them as they are.

7.1 A Hybrid Diagram Example

A Multi-Tier Client Server System is one, in which every Component is assigned a *Tier*. A Tier is similar to a Layer, but the difference is, it groups Runtime entities and not implementation units. Here the Client is in first Tier. The Web Server is in the Second Tier, and the Database is in the Third Tier. It is clear that none of the existing Styles can be used to depict this type of Architecture. However, we can use the Hybrid Diagram, to document such architectures. shows precisely, how we can do that. We can import two instances of Client/Server Style, and use Generic Component and Generic Connections to do the same.

7.2 Example Custom Styles

We now present some more example of usage of Hybrid Diagrams, this time specifically to produce new Custom Styles. As we have already mentioned, there are a number of Styles known today, both C&C, as well as Module Styles.

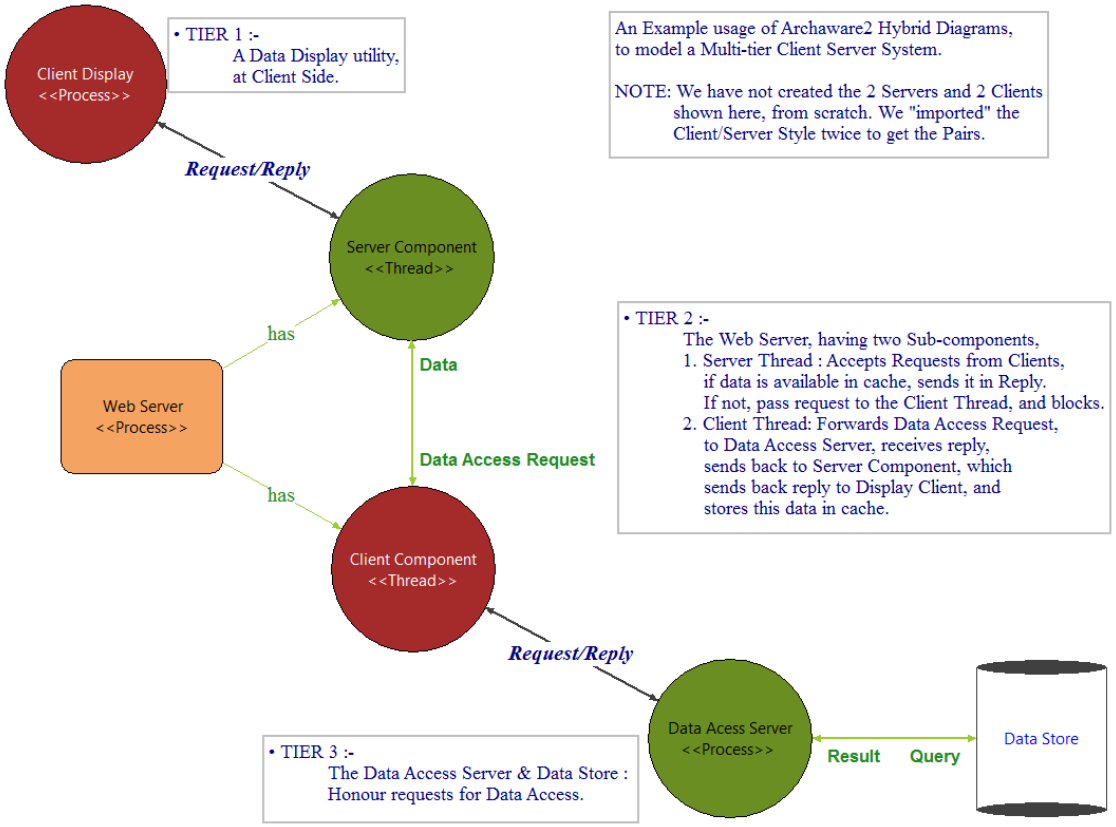

Figure 7.1: An Example showing implementation of *Multi-Tier Client Server System* using Hybrid Diagrams

Archaware2 currently supports only 6 of them. Some more Styles, which are not a part of Archaware2, are documented using the Hybrid Diagram, and then exported, as a Custom Style. This section browse through these styles. Figure 7.2 to Figure 7.6 show these Custom Styles.

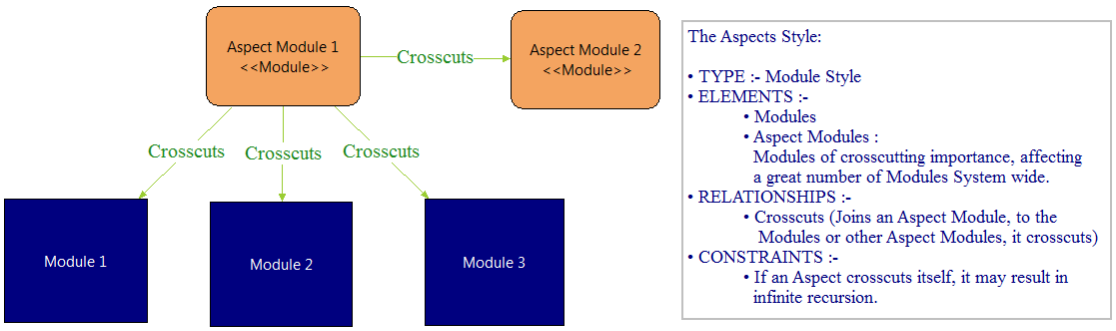

Figure 7.2: An Example of Custom Styles: The *Aspects Style*

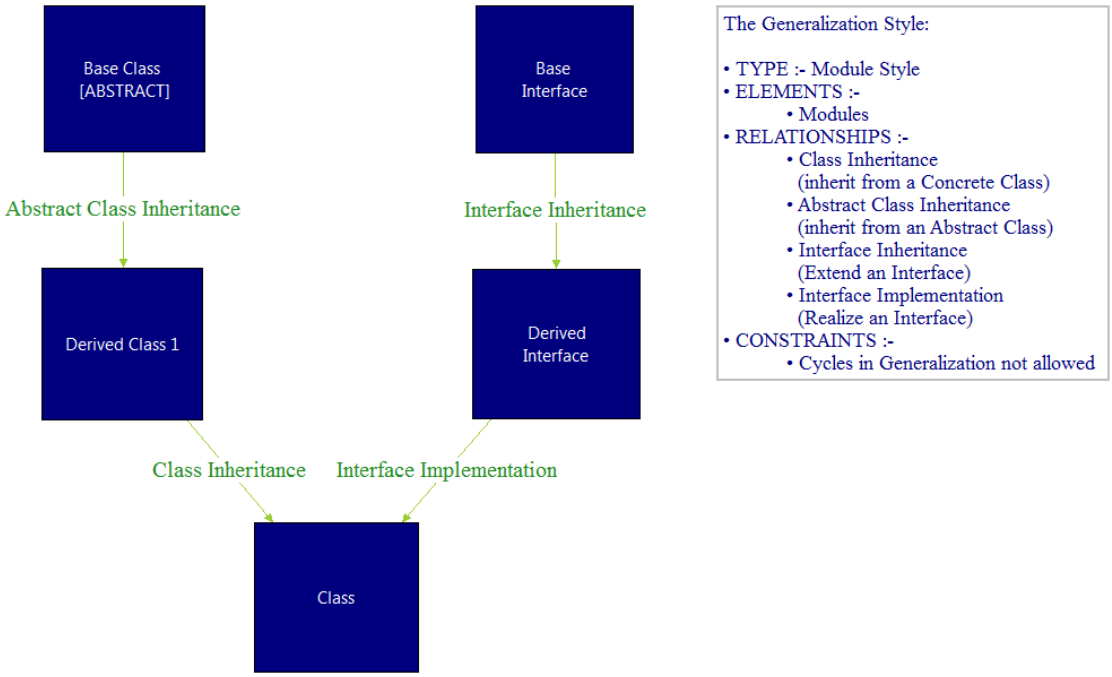

Figure 7.3: An Example of Custom Styles: The *Generalization Style*

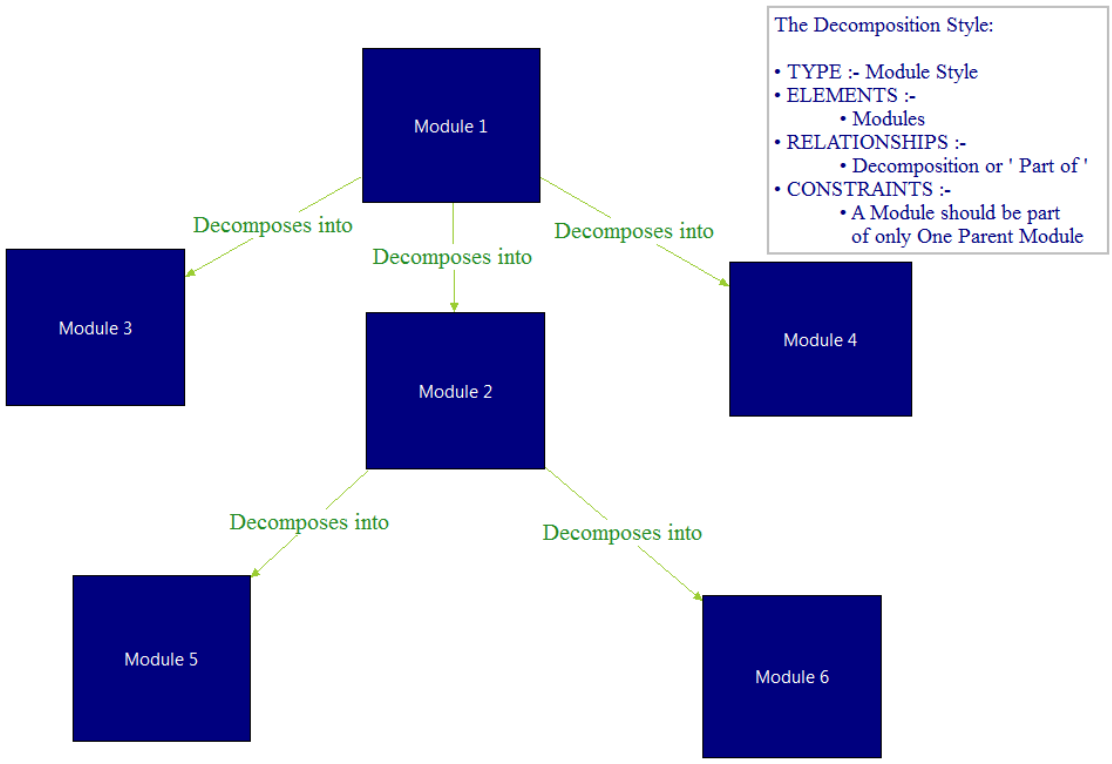

Figure 7.4: An Example of Custom Styles: The *Decomposition Style*

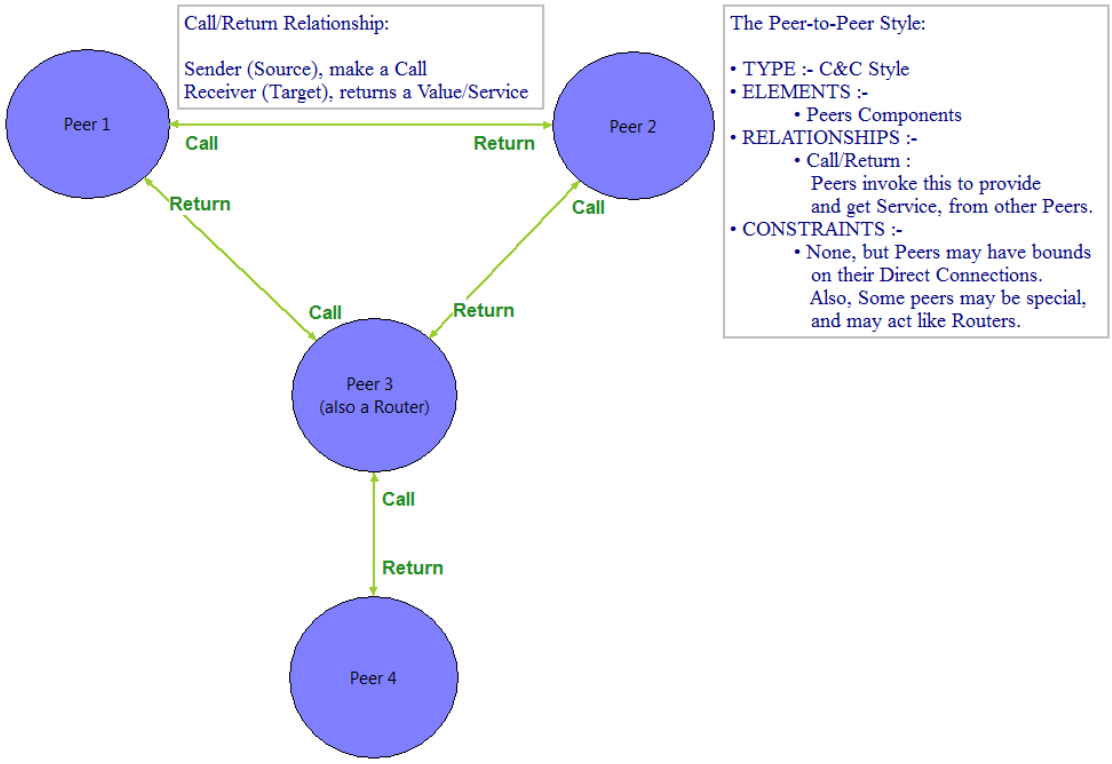

Figure 7.5: An Example of Custom Styles: The *Peer-To-Peer Style*

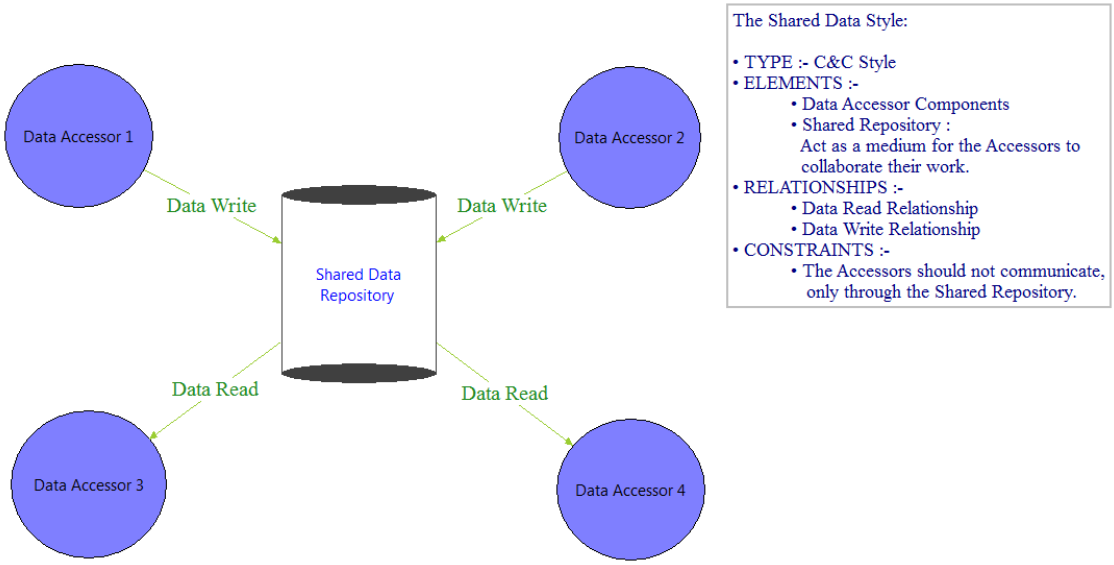

Figure 7.6: An Example of Custom Styles: The *Shared Data Style*

Chapter 8

Conclusion and Future Work

As a *Conclusion*, we can say, Archaware2 is a Software Architecture Documentation Tool, with a perception of ease to use. The tool provides click-and-draw interface for drawing Architecture Diagrams, without bugging the user for intricate details of the same. The User can import and extend, any of the supported styles, or may chose to craft a style on his own, and use it later. The tool's architecture is strongly in line with the Architecture of the Graphical Editing Framework, making it easy to extend by future developers.

There can be a number of *Future Enhancements* that can be incorporated in Archaware2. We list a few of them here:

1. New Styles can be added other than the 6 styles already supported.

2. The User can be given Preferences on choosing Background and Foreground Colors for the Elements and Connections.

3. More Elements and Connections (which may have cross style application) can be added in the Hybrid View.

4. The Diagrams can be exported in an XML based format, which can be read and modified using any XML editor.

Bibliography

[1] Software Engineering Institute. Software architecture defenitions. http://www.sei.cmu.edu/architecture/start/community.cfm.

[2] Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17:40–52, October 1992.

[3] Philippe Kruchten. The 4+1 view model of architecture. *IEEE Softw.*, 12:42–50, November 1995.

[4] Philippe Kruchten, Mary Shaw, Grady Booch, Rich Reitman, and Kurt Bittner. http://217.126.172.252/WebTecnica/Programacion/Objetos/02UML/Referencias/02Presentaciones/arch.pdf.

[5] University of California. Archstudio 4 website. http://www.isr.uci.edu/projects/archstudio/.

[6] IBM Corporation. Rational rose website. http://www-01.ibm.com/software/awdtools/developer/rose.

[7] Carnegie Mellon University. Acme studio overview. http://www.cs.cmu.edu/~acme/docs/language_overview.html.

[8] Eclipse Foundation. Eclipse website. http://www.eclipse.org/org.

[9] Gef developer's guide. http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.gef.doc.isv/guide/guide.html.

[10] Draw2d developer's guide. http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.draw2d.doc.isv/guide/guide.html.

[11] Swt website. http://www.eclipse.org/swt/.

[12] Eclipse corner article: A shape diagram editor. http://www.eclipse.org/articles/Article-GEF-diagram-editor/shape.html.

[13] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond (2nd Edition)*. Addison-Wesley Professional, 2010.