Developing a Modeling Tool Using Eclipse

Nick Kirtley, Ahmad Waqas Kamal, Paris Avgeriou

Department of Mathematics and Computer Science
University of Groningen, the Netherlands
n.e.kirtley@student.rug.nl, a.w.kamal@rug.nl, paris@cs.rug.nl

Abstract

Tool development using an open source platform provides autonomy to users to change, use, and develop cost-effective software with freedom from licensing requirements. However, open source tool development poses a number of challenges, such as poor documentation and continuous evolution. In this paper, we present our experience of developing a modeling tool in an open source environment. We not only explain the functionality of the tool, but also list the pros and cons of developing in such an environment. The contribution of this paper lies in introducing readers to the modeling tool and draw attention to some of the challenges faced by open source tool developers.

Keywords: Open Source; Eclipse; UML; Modeling; Architectural Patterns

1 Introduction

During the past few years, using open source environments for tool development has become a widely known paradigm to develop new software [2]. The research projects carried out by different research institutions, which often have budget and resource restrictions, can benefit from such environments to provide a practical implementation of their research work. Currently these open source platforms cover a wide range of software disciplines e.g. Eclipse MDT [3] provides a modeling framework, Apache [4] provides web-engine support, and ArgoUML [5] is used for modeling software in UML etc. Such open source environments provide the user with a base of technology and an expert online community of developers to share their experiences in order to exploit ways to develop software that best meets the requirements at hand.

However, using an open source framework such as Eclipse has its advantages and disadvantages. On the one side, Eclipse offers a highly extensible platform, an active community of online developers, and a large number of plug-ins that can be further customized to meet the specific needs of the problems at hand. On the other side, it lacks explicit documentation support and carries an evolving underlying platform.

In our previous work, we have discovered a set of architectural primitives that are recurring abstractions found among a number of architectural patterns [1]. One benefit these primitives offer is their use as basic building blocks for modeling patterns. However, such a novel research direction needs a practical implementation with adequate tool support to allow users to systematically model primitives in software design. Moreover, we describe how the existing tools, such as ACME Studio [6], Wright [7], and ArgoUML [8] lack considerably in providing such support, thus requiring either an extension mechanism to extend one of the existing tools or creating a new tool from scratch.

In this paper, we use the open source Eclipse framework, which provides a rich technology base to create plug-ins as an extension to its underlying platform. We use the UML2 metamodel using the APIs provided in the Eclipse UML plug-in and create primitive-specific profiles that can be applied to the UML model. The tool, that we have called Primus, allows users to systematically model architectural primitives with the eventual goal of modeling software patterns and architecture. UML elements, used for defining the primitives, are further validated by the use of the Object Constraint Language (OCL) [15]. This not only helps to validate the model but also helps in finding and locating the primitives modeled in software design. The main contribution of this paper lies in introducing the research

community to the Primus modeling tool, and our experiences of developing tool in the Eclipse modeling framework that can serve as guidelines for researchers and developers that wish to develop similar modeling tools.

The remainder of this paper is structured as follows: in Section 2, we present the research work background that provides a base for Primus. Section 3 gives detailed information of the goals for Primus, the approach taken to develop it and we present our approach for representing primitives as modeling abstractions, exemplified using an extension of the UML. Section 4 describes the design and usage of the tool. Section 5 describes the modeling of a few selected pattern variants using primitives with the help of a non-trivial example. Section 6 discusses the lessons learnt during the development of tool using the open source environment. Section 7 describes related work and Section 8 discusses future work and concludes this paper.

2 Motivation and Research Work Background

In this section, we briefly present the notions of architectural patterns alongside our previous research work in devising an approach for the systematic modeling of architectural patterns. We motivate that the use of architectural primitives, which are recurring abstractions found among a number of patterns, can be used as a way for the systematic modeling of architectural patterns in system design.

Architecture Patterns and Architectural Primitives

Among a number of software patterns that exist in the literature, architectural patterns [9], and design patterns [10] are the most widely known and used. It is difficult to draw a clear boundary between both types of these patterns, because it depends on the way these patterns are perceived and used by software architects. The work in [9] is more concerned about architectural issues, i.e. high-level components and connectors while the work in [10] lists a number of specific solutions to design problems. In this paper, we focus on the former and in section 5; we show the modeling of some selected architectural patterns in an example software design.

In our previous work, we have discovered a number of architectural primitives in the component-connector view [1]. These primitives are found repetitively among a number of architectural patterns and serve as the basic building blocks for modeling architectural patterns in software design. The base of our work, for developing a modeling tool relies on using architectural primitives for modeling architectural patterns in system design. In the upcoming sections, we show the actual modeling of the primitives in the Primus tool while in this section; we briefly describe all the primitives documented during our previous work:

Primitive	Description
Callback	A component B invokes an operation on Component A, where Component B keeps a reference to component A – in order to call back to component A later in time.
Indirection	A mechanism that allows for redirection of a message/communication to a target component with the purpose of hiding the location of the target component.
Grouping	Grouping represents a Whole-Part structure where one or more semantically coherent components work as a Whole while other components are its parts.
Layering	Layering extends the Grouping primitive, and the participating components follow certain rules, such as the restriction not to bypass lower layer components.
Aggregation Cascade	A composite component is composed of a number of subparts, and there is the constraint that composite A can only aggregate components of type B, B only C, etc.
Composition Cascade	A Composition Cascade extends Aggregation Cascade by the further constraint that a component can only be part of one composite at any time.
Shield	Shield components protect other components from direct access by an external client. The protected components can only be accessed through an intermediary component.
Typing	Custom typing models are defined using with the notion of super type connectors and type connectors.
Virtual Connector	Virtual connectors reflect indirect communication links among components for which at least one additional path exists from the source to the target component.

Push-Pull	Push, Pull, and Push-Pull structures are common abstractions in many software patterns. They occur when a target component receives a message on behalf of a source component (Push), or when a receiver receives information by generating a request (Pull). Both structures can also occur together at the same time (Push-Pull).
Virtual Callback	Consider two components connected via a callback mechanism. In many cases, the callback between components does not exist directly, rather the callback operation takes place through one or more mediator components.
Adaptor	This primitive converts the provided interface of a component into the interface that the clients expect.
Passive Element	Consider an element that is invoked by other elements to perform certain operations. Passive elements do not call operations of other elements.
Control	Calling a method in the target component can involve passing a control from source to the target component.
Mediator	Sometimes certain objects in the set of objects cooperate with several other objects. Allowing a direct link between such objects can overly complicate the communication and result in strong coupling between objects [11]. To solve this problem, mediator components are used.

Table 1: A brief description of Primitives

3 Tool Development Goal and Approach

In this section, we describe our motive of tool development, rationale behind the selection of Eclipse open source platform, and the mechanism to extend UML for defining primitives.

3.1 Goal

The theory of architectural primitives described in the previous sections needs a practical implementation, in the form of a tool, which fulfills at least the following requirements:

- To provide reusability and visual modeling support to express architectural primitives
- To provide automated model checking support in order to validate that the primitives are correctly applied in software design
- To facilitate the extensibility of the current set of primitives with the definition of newly discovered architectural primitives
- To support the customization of the provided primitives in order to meet the specific needs of the problem at hand

3.2 Tool Development Platform Selection

The requirements listed above demand, as explained below, the usage and extension of a platform and modeling language that is extensible enough to capture the semantics of the primitives. We take into consideration the following modeling languages and platforms to advocate the selection of the Eclipse open source platform, as follows:

ACME: ACME provides a template mechanism, which can be used for abstracting common reusable architectural idioms and patterns [12]. ACME allows defining user specified constraints on architecture elements to model patterns. Violations of these constraints are automatically checked in ACME studio [6]. However, ACME lacks in documentation and there is virtually no active community to discuss the development issues. Due to the lack of documentation, it is difficult to determine the structure of the underlying meta-model. However, ACME scores high in adaptability and reusability. The reusability is the main advantage of ACME. ACME is a close match with our project but the serious lack of documentation means that we would be very dependent on the ACME development team.

Eclipse Model Development Tools (MDT): The Eclipse MDT project [3] is very large and so is the user base. This means that the community support is excellent and a fair amount of documentation is available. MDT is obviously modeling based so the match is very close to Primus. In fact the meta-model used is based on the UML 2.1 metamodel. Another added advantage of MDT is that it provides a wide range of plug-ins; some of these plug-ins provide graphical support for modeling UML diagrams, which can be further extended to fulfill the

software requirements. Thus, the Eclipse MDT has excellent support and the reusability is high due to the model-based features.

Argo UML: Argo has good documentation but it is not yet very stable. The meta-model is not thoroughly documented and there have been many reports of slow fix rate of bugs. Considering these issues, we consider Argo UML as a weak option to develop the Primus tool.

Archium: Archium is an ADL based on documenting the design decisions [18]. Although, the technical documentation related to the tool is poor, it does have the advantage that it was developed locally in our department and thus questions can be answered in person.

Conclusion: The Eclipse MDT is an open source project and it has the advantage that all source code is available online and can be openly distributed upon change. Eclipse provides development support in the form of forums and this is by far the best way to learn about what Eclipse MDT is capable of and how to extend it in any way. This support is far better than support offered from other modeling languages described above. The MDT project has a number of plug-ins and the ones relevant to our tool development include – UML2, OCL, and UML2Tools. The UML2 plug-in offers UML 2 metamodel support, the OCL plug-in offers OCL model checking support and the UML2Tools plug-in offers visualization support. This clear separation of functionality has the advantage that it is clear what part of the package needs to be used for the relevant functionality of the project.

Another advantage of the Eclipse framework is the fact that it has the UML2 plug-in that offers the UML 2.1 metamodel. The fact that the UML2 plug-in is based on the UML 2.1 metamodel is a huge advantage because it means that any modelers using the Primus system will know exactly what the metamodel consists of because of the very detailed specification supplied by Object Management Group (OMG) [13].

3.3 Extending UML to define Primitives

As described above, an additional benefit that we achieve with the selection of Eclipse modeling framework is its support for UML, which is widely known as an industry standard modeling language and is highly extensible [14]. There are two approaches to extend UML: extending the core UML metamodel or creating profiles which extend metaclasses. Our work focuses on the second approach where we create profiles specific to the individual architectural primitives. That is, we define the primitive as extensions of existing metaclasses of the UML using stereotypes, tagged values, and constraints:

Stereotypes: Stereotypes are one of the extension mechanisms to extend UML metaclasses. We use stereotypes to extend the properties of existing UML metaclasses.

Constraints: We use OCL to place additional semantic restrictions on extended UML elements. For instance, constraints can be defined on associations between components, navigability, direction of communication, etc.

Tagged Values allow one to associate tags to architectural elements. For example, tags can be defined to represent individual layers in a layered architecture using layer numbers.

4. The Primus Tool

The Primus tool has been developed to provide a practical implementation of architectural primitives. The tool interacts with the UML component diagram by allowing the user to add primitives to the model and for model checking capabilities. The architecture, functionality, and usage of the tool are described in the remainder of this paragraph.

4.1 Primus Functionality

The Primus tool is mainly intended to assist software designers in systematically modeling architectural primitives in system design. The two main areas of functionality of Primus are: 1) modeling primitives and 2) model checking of primitives. The tool allows for primitives to be applied to existing UML elements in the Component-Connector view or for primitives to be applied with all the necessary UML elements. For example, a Callback primitive consists of two components that are connected. It is possible to apply the Callback primitive to two existing components or to create new components automatically. This option allows multiple primitives to be applied to the same component. Adding primitives is achieved via a wizard whereby the wizard is opened via a context menu. A wizard offers the user with a step-by-step explanation of the possible choices and thus requires a smaller learning curve for the user.

Model checking/validation of primitives in the Component-Connector view consists of 1) finding primitives in a UML model and 2) validating the primitives found and indicating any problems. Finding primitives in a model involves Primus returning the relevant UML elements that are part of a primitive. For example, if we have a model with ten components and a Callback applied between two components then the result will be the two components that are part of the primitive. Model checking the primitives found consists of checking whether all aspects of the primitives have been applied properly and whether any primitive specific rules have been broken. The results also include problem specific information.

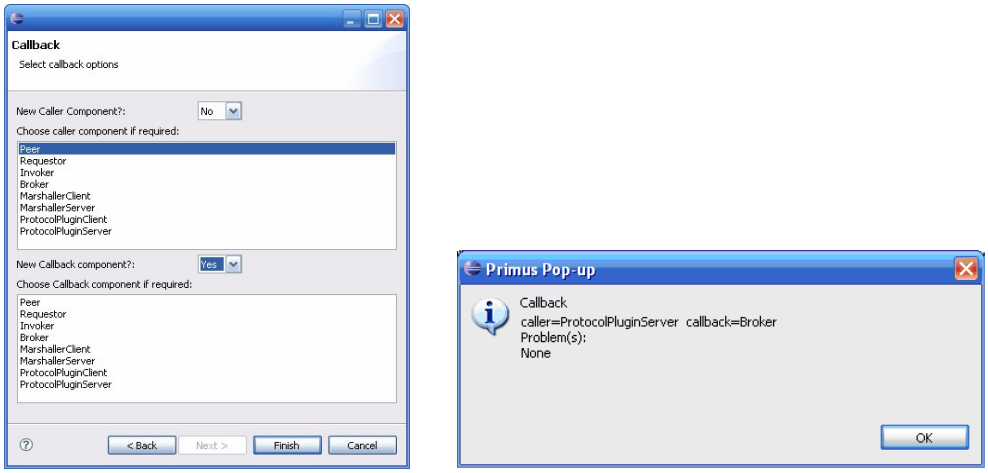

Figure 1: Wizard and Validation Information

Figure 3 shows two screenshots from Primus. The left screenshot shows some of the options from the primitive modeling wizard and the right screenshot shows the result of model checking/validation for the Callback primitive.

4.2 Challenges

One of the main challenges during this project has been the use of OCL for finding (in the model) and validating primitives while also returning relevant error information if a primitive has been applied incorrectly. The reason we chose to use OCL (Rather than Java) is because OCL queries are relatively small and can be understood more easily by those that are unfamiliar with the details of the project and users that wish to implement newly found primitives. An example code segment can be found in the next section.

4.3 Tool Architecture

The Eclipse framework consists of a plug-in architecture. Therefore, developing a modeling tool is performed by means of a plug-in that extends the existing framework. The plug-ins that we have used and extended are from the Modeling Development Tools project

[3]. The plug-ins of interest from the MDT project are UML2, OCL and UML2Tools. The UML2 plug-in provides the UML 2.1 metamodel and the ability to define UML 2.1 models, the OCL plug-in provides OCL querying and constraint capabilities and the UML2Tools plug-in provides UML diagram interaction with the UML 2.1 models.

The Primus tool is a fragment plug-in that extends the component diagram plug-in from the UML2Tools plug-in. A fragment plug-in is a type of plug-in that merges the host plug-in with a newly developed plug-in. We chose to use a fragment plug-in because the interaction between the user and the plug-in is from the component diagram. The plug-ins UML2 and OCL are used by Primus.

The Primus plug-in follows the event driven style: all functionality starts due to an action from the user. Once an event is triggered, it is passed to the relevant classes to carry out the necessary action.

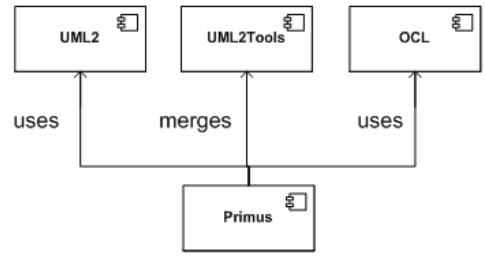

Figure 2: Direct plug-in dependencies

Figure 2 shows the direct dependencies of Primus while Figure 3 shows the class diagram of Primus. The event driven design of the system is used and the relationships between the classes in the diagram show the two main areas of functionality of the system: primitive modeling and model checking. The core implementation of the project is contained within the CreatePrimitive, OCLCheck and PrimitiveRegister classes. The CreatePrimitive class deals with the necessary primitive modeling aspects based on user input, the OCLCheck class contains queries and performs them, and the PrimitiveRegister class interprets the OCL query results. Moreover, the inheritance relationship shown between the three UML packages and the Primus classes show that the Primus classes are inheriting the functions one or more classes of these packages.

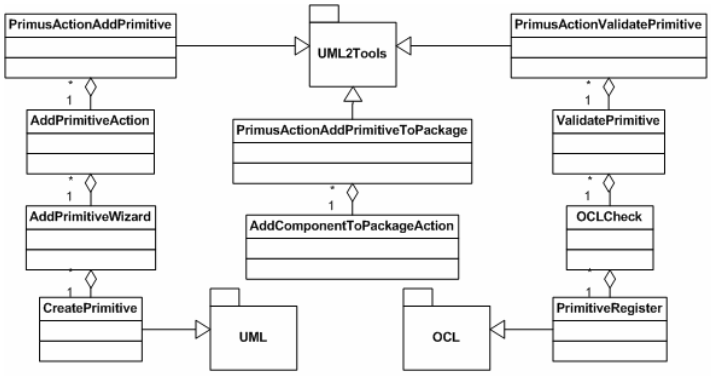

Figure 3: Class diagram with dependencies

4.4 The Eclipse Plug-ins

The eclipse platform provides support to create plug-ins, which can later be merged in the eclipse environment, thus providing a network of plug-ins that can meet the specific development needs of the project at hand. In the previous sub-section, we have listed the plug-ins that collaborate with the Primus plug-in. In this section, we describe the usage of these

plug-ins in the context of the Primus plug-in and elaborate on the functions supported by these plug-ins.

– *UML2 Tools*

The UML2 tools project is a set of Graphical Modeling Frameworks (GMFs) for visually modeling UML diagrams. Eclipse UML2 tools provide support for nine different kinds of UML diagrams namely Class, Profile Definition, Component, Activity, State Machine, Composite Structures, Deployment and Use Case [3]. In this work, our focus lies on using only the component diagram package available in UML2 tools. The UML2 tools plug-in allow to create UML models both programmatically and by using the UML editors. Moreover, the plug-in allows to work with UML profiles, create UML models, and customize UML metamodel elements. The Primus uses the UML2 tools editor to create primitives specific profiles to work with UML models using both the existing UML model elements and the UML elements specifically customized for modeling primitives.

– *The Object Constraint Language*

The Eclipse OCL plug-in provides an API for parsing and evaluating OCL constraints and queries on EMF model. In the case of UML2 tools, the UML models are constrained and checked by using OCL. We do not extend the semantics of the OCL parser and only use the OCL queries to locate the primitives and identify missing primitive elements in the UML model.

5. An Example of Modeling Patterns Using Primitives in Primus

The Primus tool is able to model patterns using primitives. This will be shown by modeling an example system called Leela as described in [1]. Due to space restrictions, we will not include the whole design in this section.

The simplified Leela system used is a layered peer-to-peer system with a broker to facilitate communication between peers. The purpose of this architecture is for a peer to communicate with another peer. To this end, the peer component communicates with components in lower layers where the Broker component connects the two peers. To model the Layers pattern we will use the Layers primitive. A Broker consists of a client side Requestor and a server side Invoker. The Requestor and Invoker require a Marshaller for communication. The client and server side Marshallers are protected using the Shield primitive. The Requestor and Invoker components are connected via the Broker component. This connection is modeled using the Virtual Connector primitive. The ProtocolPlugin components are used to provide the necessary protocols for the Broker component. The server side of the ProtocolPlugin has a Callback applied so that the Broker component has a reference to the ProtocolPlugin component.

Note that the system modeled is only a demonstration of the primitives and not the system itself.

Figure 4 shows the Leela system modeled in Primus. Some tags have been added to simplify reader interpretation.

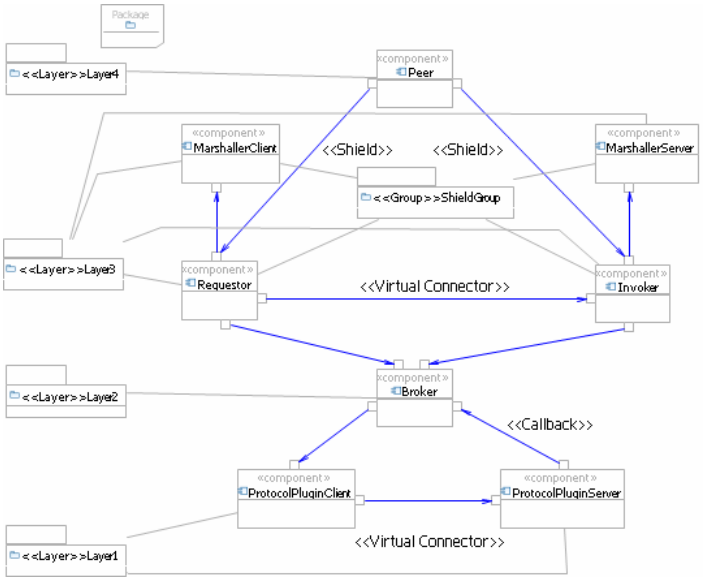

Figure 4: Simplified Leela system

We will also present a validation example for the Callback primitive. If the Callback primitive has been implemented properly, the feedback to the user will something similar to Figure 1. This means that the model-checking feature has found a Callback and that it has been found between the components stated. If we now change a critical part of the Callback then it should also inform the user of the exact nature of the problem. For example, an essential part of the Callback primitive is the stereotyping of one of the interfaces with the IEvent stereotype. If we undo the application of the stereotype the model checker will return the message from Figure 5.

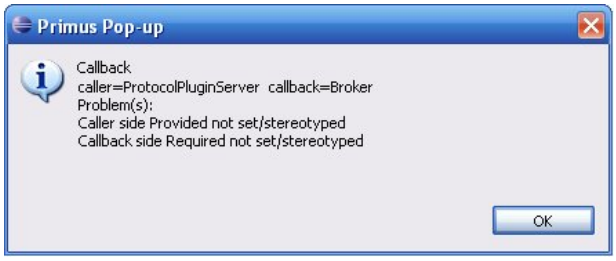

Figure 5: Callback model checking with error

This message shows that the model checker is able to find a primitive that has not been applied properly and is able to state the exact problem. Therefore the user can easily correct the problem.

Using OCL to achieve the goals stated above has been achieved by returning every sub-result needed to validate a primitive. If a sub-result differs in value from the expected value then an error has been found and we automatically know what the error is. A further interpretation of the results is needed in Java so that the result can be presented properly to the user.

```
Component.allInstances()->collect(i |
       i.ownedConnector->collect(conn |
       let callerPortSter : Port = conn.end.partWithPort->
              any(owner=i).oclAsType(Port)->
              any(p | p.oclAsType(Port).getAppliedStereotypes()->
              any(name='EventPort')-> notEmpty()),
                     callbackPortSter : Port =conn.end.partWithPort->
              any(owner<>i).oclAsType(Port)->
              any(p | p.oclAsType(Port).getAppliedStereotypes()->
              any(name='CallbackPort')-> notEmpty()),
Tuple{c1=i,c2=otherComp,callerPort=callerPortSter,callbackPort = callbackPortSter
```

The code above shows a segment of the Callback query. The callerportSter and callbackPortSter are sub-results that we are interested in and if a null value is returned we know that the relevant port was not stereotyped. This query will of course return many potential Callbacks within the model. The query can then be filtered to only include potential Callbacks that have x or less amount of problems.

6. Lessons Learnt

The Primus tool was developed to investigate the practicality of the research work carried out by us. The results, for the systematic modeling of architectural primitives in system design, are thus far quite encouraging. Based on our experience, we are quite capable to draw some important lessons that we learnt during the development of Primus tool in the open source Eclipse environment, as described below:

Complexity: Eclipse UML tools' API support for the visual modeling of software in Component-Connector view is still under development. In the absence of graphical editors for modeling architectural elements in software design, a developer has to interact directly with the UML elements in its metamodel in Component-Connector view. Although, the Eclipse UML2 tools community is working to provide visual features for modeling software in Component-Connector view, the current plug-in is hard to use and understand with the existing layout. This introduces the complexity to interact with the Eclipse UML tools plug-in in the Component-Connector View.

Constant Development: The Eclipse MDT provides a solid base for plug-ins development with the purpose of a system where different plug-ins can interact with each other to provide better reusability and efficient development support. In the case of Primus tool, the plug-ins that we interact with are UML2 tools and OCL. However, being immature open source modules, these plug-ins are in constant development resulting in an evolving state, which is difficult to manage and understand.

Lack of Documentation: Currently, the documentation concentrates on how to use the plug-ins with some basic examples. This proved to be a big challenge during the development of the Primus tool, which involves a complicated interaction with Eclipse plug-ins through a number of interfaces. We found the following three challenges during the development of the Primus tool:

- There are not enough coding examples documented by the Eclipse community
- There is not enough technical information about how the plug-ins has been developed.
- The general architecture of the plug-in is hard to understand in the absence of design documents

Poor OCL editor: We use OCL queries to locate the primitives in the system design and to return an error value when a primitive is incorrectly modeled in system design. However, the OCL queries, that return an error, can lack detail in the explanation and location of the error. An ideal situation would be the exact nature of the error, what line of code generated the error and at what point in the model generated the error. A more advanced editor could help in developing queries. Perhaps similar to SQL query editors.

Performance Issues in Eclipse: Eclipse poses high requirements on the hardware front on which the system requires adequate free memory and high processor speed. This at times results in abnormal termination of program or deadlock situation, even when a slightly less capable system is used as compared to the deemed system requirements.

Time and Cost vs. Learning Curve: We consider the following factors that consumed most of our time while working with the Eclipse open source environment:

- *Installation*: In the absence of concrete guidelines for the installation of Eclipse MDT along with the necessary plug-ins, the correct installation and setup of Eclipse platform

consumed extra time and effort. We spent nearly 3 weeks to install and configure the Eclipse environment.

- *Understanding the Eclipse UML tools*: Being new to the Eclipse environment, most of the Eclipse UML tools understanding was done using trial and error approach, which consumed much of the time. The Component-Connector diagram of UML is still in the development phase and considerably lacks in providing adequate information about how the UML specification maps to the Eclipse UML plug-in. Certain aspects of the UML specification are implemented in a very specialized way in the Eclipse environment. For example the connector metaclass is represented in a specific way in the Eclipse UML plug-in, making it difficult for users to understand it without the proper documentation.
- *Online Community*: The most positive aspect of using the Eclipse MDT was its active and responsive online community that shared their experiences and provided us the technical feedback to our queries using online forums. This not only helped us in using the Eclipse plug-ins and its platforms but also it provided us the opportunity to report bugs and hence got them fixed either by the regular Eclipse developers or even by the user itself.

7. Related Work

The Primus tool development using the Eclipse environment described in this paper provides a practical implementation of our previous work [1] where we present a set of primitives for modeling architectural patterns. The use of the Eclipse environment and work with its plug-ins is not novel as a large community of developers and users is already actively involved in working with the Eclipse environment. However, the novelty of our work lies in documenting the challenges associated in using the Eclipse environment from the development perspective.

The related work to our work is two fold in this perspective: a) the developers using the Eclipse open-source environment; and b) the experiences documented by researchers to investigate the suitability of using open source platforms. However, as per our knowledge, there exist little documented information in using open source environments in general and Eclipse in particular. Despite the presence of some of the tools developed using Eclipse (e.g. Visual Paradigm [19]), there has been a little focus in documenting the challenges. Few other researchers [2] [16] have documented some of the challenges.

Michelle [16] highlights the challenges related to user interface, documentation, programming, etc. in open source environments. He highlights the cost vs. ease of use when using the open source environment. Our work distinctively differs from him as we work with a tool which involve more depth interaction with the Eclipse open source environment which lets us document more solid lessons for using such platforms.

Simon et al. [17] extends the UML metamodel by creating pattern-specific profiles. The work by Simon et al. maps the MidArch ADL to the UML metamodel for describing patterns in software design. However, this approach does not address the issue of modeling a variety of patterns documented in the literature. Therefore, rather manual work is required to create profiles for each newly discovered pattern. Our approach distinctively differs from this work as we focus on describing a generalized list of patterns using the primitives.

The work by Dim et. al. [20] propose a re-factoring approach that uses the Reba tool to connect mismatched software libraries. However, their approach is more helpful in the plug-in development where the base functionality of a pug-in is updated leaving it in a state of mismatch with the collaborating plug-ins. Our approach is different because we use only the interfaces of existing eclipse plug-ins without affecting their internal functionality. We expect that the interfaces offered by the Eclipse plug-ins will not be affected with the further development of existing plug-ins.

8. Conclusion and Further Work

This paper describes our experience of developing the Primus modeling tool using the Eclipse open source environment. The development experience not only provides a practical implementation of our ongoing research work but it reveals many challenges and benefits of using the Eclipse platform for the tool development. During the process, we have found that the Eclipse MDT provides good support to re-use existing plug-ins and rich technical support from online communities. However, it comes up with numerous challenges such as poor documentation, constant development of plug-ins, and weak visualization support for UML component diagrams due to immature state of relevant plug-in.

We may conclude that, relating to tool development using Eclipse environment, Eclipse plug-ins still lack maturity. In fact, the significant aspect of underlying Eclipse platform documentation is missing and the only way to understand the Eclipse MDT domain is through the online community support. However, we do not consider the online community as an alternative to thorough documentation especially when it comes to the understanding of Eclipse MDT design. However, the online community is highly beneficial in sharing experiences and devising an approach that best meets the problem at hand.

We have concentrated on the initial step of systematically modeling and finding the architectural primitives in system design. The results generated so far are quite satisfactory and provide a solid base for further extending the tool for the systematic modeling of architectural patterns. Moreover, eventually we plan to cover software modeling in other views such as the behavioral view.

References

[1] Uwe Zdun and Paris Avgeriou. Modeling Architecture Patterns using Architecture Primitives, OOPSLA' 05, ACM, October 2005

[2] Diomidis Spinellis, Clemens Szyperski,, IEEE Software, 0740-7459, Copyright 2004, IEEE

[3] Model Development Tools, www.Eclipse.org/mdt

[4] The Apache Software Foundation, http://www.apache.org

[5] ArgoUML, http://argouml.tigris.org

[6] ACME Studio, http://www.cs.cmu.edu/~acme/AcmeStudio/tutorials.html

[7] Robert Allen and David Garlan, A Formal Basis For Architectural Connection, ACM Transactions on Software Engineering and Methodology, vol. 6, no. 3, pp. 213-249, July 1997

[8] Borger, Jrobbins, Linus, A UML design tool with cognitive support, http://argouml.tigris.org

[9] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt, Pattern-Oriented Software Architecture: On Patterns and Pattern Languages, John Wiley & Sons, ISBN 978-0-471-48648-0

[10] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns: Elements of Reusable Object Oriented Software, Addison-Wesley Professional Computing Series, 1995

[11] Frank Buschmann, Kevlin Henney, Douglas C. Schmidt, Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing, John Wiley & Sons, ISBN 978-0-470-05902-9

[12] David Garlan, Robert Monroe, David Wile, ACME: An Architecture Description Interchange Language, Proceedings of CASCON 97, Toronto, Ontario, pp. 169-183, January 1997

[13] OMG, , 2007, Model Driven Architecture, http://www.omg.org/mda/

[14] Nenad Medvidovic, David S. Rosenblum, David F. Redmiles, Jason E. Robbins, Modeling Software Architectures in the unified modeling language, ACM Transactions on Software Engineering and Methodology, vol. 11, no. 1, pp. 2-57, January 2002

[15] Object Constraint Language Specification versions 1.1, OMG standard, http://umlcenter.visual-paradigm.com/umlresources/obje_11.pdf

[16] Michelle Levesquem, http://www.firstmonday.org/issues/issue9_4/levesque, March 1 2004

[17] Simon Giesecke, Florian Marwede, Matthias Rohr, Willhelm Hasselbring, A Style-Based Architecture Modeling Approach For UML2 Component Diagrams, In Proceedings of Software Engineering and Applications, SEA 2007, Cambridge, MA, USA, 2007

[18] Jansen, A. van der Ven, J. Avgeriou, P. Hammer, D. K, Tool Support for Architecting Decisions, Working IEEE conference, WICSA 2007

[19] Visual Paradigm for UML, http://www.visual-paradigm.com/product/vpuml

[20] Danny Dig, Stas Negara, Ralph Johnson, Vibhu Mohindra, ReBA: Refactoring-aware Binary Adaptation of Evolving Libraries, ICSE'08, May 10–18, 2008, Leipzig, Germany