Model Refactoring for Software Architecture using UML

R. Aroul canessane, Dr. S. Srinivasan

Abstract— Software architecture provides a high level abstraction of the software systems. The software architecture research provides a better system by modeling the core aspects in the system development. Various Architectural Design languages (ADL), tools, styles and technologies have emerged due to the undergoing research’s in software architectures. The architectural design research has been basically classified into two distinct groups. 1. The academicians primarily focus on the architectural descriptions analytical evolution and 2. The industry people they choose a model and check out the issues that arise while developing the software models. Both the academicians and the architectural design group’s lack in both (a) forward engineering and (b) reverse engineering. A forward engineering is refining a high-level abstraction of a software system model into a low level model and the reverse engineering is a low-level to a high level abstraction model. This paper describes the design of our experience in using UML which is an object oriented design language. This paper explains the suitability of UML for modeling the architectural concepts. We have provided a framework which has an extension of UML which identifies and resolves mismatches within the model and also across the different UML views. This paper discusses the issues at the same level and across level of abstraction as well as the approaches adopted to resolve those issues using our current tool support.

Keywords—connectors, re-engineering, reverse-engineering, transformation, views.

I. INTRODUCTION

SOFTWARE architecture provides high level abstractions which results in better software system architecture in the early stage of the development. Choosing the appropriate aspect to model and evaluate the performance are the two decisions that has to be framed, which is under software architecture research. One part of the software architecture community which is under research is focused on the analytical evolution of architecture. More number of ADL’s are proposed and these ADL’s has their own specific approach for specifying and evolution of the software architectures, that depends upon the aspects of the specific system in depth.

The second approach focus on the modeling issues [1] which arise while developing the software, most of the problems and the errors cannot be detected easily.

TABLE 1.

<table>
<thead>
<tr>
<th>Academic people approach</th>
<th>Industry people approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus on the analytical quality of architectural models</td>
<td>Focus on the architectural development issues</td>
</tr>
<tr>
<td>Individual model</td>
<td>Family of models and relate those issues</td>
</tr>
<tr>
<td>Powerful technique for analysis</td>
<td>Deal as a big picture on development</td>
</tr>
<tr>
<td>Special solutions</td>
<td>General solutions</td>
</tr>
</tbody>
</table>

The two communities who address software architecture are representend in the Table 1. The relationship with the software architecture and complexities over view are shown. In this paper we have used extended UML to span both the communities.

We have used UML because a large set of predefined constructs which supports extensibility is present in UML [2] also it supports some tools which we have used while developing some architecture. Our work concentrates on a framework which identifies and resolves the mismatch within the model and also across the different models which has different levels of abstraction (same level and across level). In this paper we have discussed about three different possible approaches using UML to model the software architectures, next we have discussed above the view point integration framework which supports the systems integrity using automated validation. Finally we have given some of the general observations by using UML for modeling the software architectures and ensure consistent, complete refinement of
the architectures to design and a reverse engineering which helps architecture from design.

II. MODELING OF SOFTWARE ARCHITECTURE USING UML

The Meta modeling architecture with four layers of UML suggests three different strategies in modeling the software architectures those strategies are

- Using existing UML strategies.
- Using the UML extensions which are built-in with all the constraints of the UML metamodel.
- Using the metamodel [3] of UML directly in the architecture modeling.

We have used the diagram which implies the four layered meta modeling architecture fig 2. These three approaches has their own advantages and disadvantages with the potential for the forward and the reverse engineering, which has been discussed below in the following strategic approach in modeling the software architecture.

A. Using the UML strategy

The simplest designing strategy is to use already existing UML model for representing the software architecture as in fig 2. The advantage of this approach is it gives an architecture model which can be easily understandable by all the UML users and can be manipulated using the UML tools that are present.

This approach does not provide any explicit relationship between the existing constructs of UML and architectural concepts; there are lots of architectural concepts missed such as connectors, components, style rules. But these relationship may encapsulated by the architect implicitly. This approach lack’s in trying the reverse engineering gin system architecture from the UML model. So far there is no UML artifact which has all the architectural information but they use some of the design elements and their interaction implicitly. Different architects create different UML model but try to implement the entire requirement which may not be helpful in reverse engineering, since different models have been created by different architects.

B. Using UML extensions with constraints

The stereotypes of UML have been addressed in the Architecture modeling which is a new concern in the software development. UML’s Meta model fig. 2b architecture is represented using the conceptualization of this approach. In this strategy the subsets of the UML are used along with the object spaces. The relevant UML part is also shown to the architects. This strategy has been applied in C2, xC2, rapid etc. The advantage of using this strategy is it deliberately represents, enforces the constraints in modeling the architecture. The architecture that has been modeled can be manipulated using the stander UML complaint tools and also can be understood by the UML users with some added effort I knowing about the stereotypes of OCL.

The drawback of this approach is it is difficult to specify the limitations or boundaries of modeling fig. 2b space. So far there is no perfect tool exist for implementing OCL constraint strategy in UML modeling. The ADL for the architecture cannot be exactly represented and implemented in UML, our study show’s this. The stereotypes which are specified by the ADL [4] are used in the UML as shown in fig. 3.

Two possibilities are there in using such ADL stereotypes. First the basic development notations of UML are used, from which some of exceptions are made using the ADL tools and second the development notation used here is UML and some of the ADL notation such as stereotypes are used to operate the specifications of UML.

C. Meta Model to UML

The Meta model which is shown in fig. 4 tempts us to use the UML for the software architecture modeling [5]. Extending this metamodel creates new capabilities in modeling into UML. The benefits of extending this Meta model, completely captures the features of all ADL’s. The advantage of using this extended version supports us to make a reverse engineering in an easier manner.
Such extension in UML needs lot of effort. To standardize this extension we have to capture all the required features without making it more compiled but in practical it leads to complexity while we make an extension. Though the extension that we have made is use full, it must be standardized as a part of UML standard, hence forth they might be compatible with UML tools and potentially create a very good architecture using the architects.

**III. ARCHITECTURAL VIEW**

Software architecture based on the ADL’s focuses on the structural constraints at the system level. Focusing only on the structural view of the model does not cover all the abstracts, because there are multiple views in designing. Part of our work is to identify the architectural mismatches that are in UML views [6]. The different views have been already spoken in our previous work fig. 5, structural, enterprise, static, configuration and behavioral view. We have created a framework for integrating the different view with a set of functions and techniques, which identifies the mismatches automatically which has been detailed in this section.

In this approach repetition of functions are used among the different view. If the functions are repeated in more than one view, that function is made as constraints in one among the views. This view integrated framework uses such constraint, which leads for consistency among the views. Along with the consistency and constraints, our framework defines what and how the information can be exchanged among the views.

The framework for the view integration is shown in fig. 4, the UML is used for designing the system model. In forward engineering, the new information’s are added to the system model and the existing views are upgraded. In reverse engineering from the existing view new high-level abstractions are used to create high level view synthesis.

When a new piece of information is added or an additional view is created it has to be validated with the system model to maintain integrity with view analysis. View analysis has the major activities such as one-The related information is mapped and it describes the overlapped information which is done with the help of naming dictionaries [7] or trace matrices. We focused our work in automating mapping using the patterns, dependencies, traces of view and shared interfaces. Two-generalize the detailed abstraction between the multiple views using transformation. Three- identify the potential mismatches that are differentiation within its elements. The automation of differentiation depends upon the mapping and transformation.

We applied our framework on the different UML views, objects, class, sequence and state chart diagram. We also extended the use of our framework in different architectural styles such as pipes and filters, layered style etc. the advantage of our framework is it can be used for both forward engineering and reverse engineering. In forward engineering we have a direct flow of detailing the architecture which is followed by design. Our framework which has view integration ensures the design consistency with the high level abstraction. With respect to our framework at first step is to check multiple views fig. 5 at various levels, in the second step we have to validate the conceptual integration of the views.
The architect has to follow the same procedure in reverse engineering. The step one is distinct when the architecture design is made. The step two validations same in both reverse engineering and re engineering. The consistency and the completion of inter-view can be validated in both the ways. Our framework which is used for integration [8] has a consistency among the views where the transformation occurs from low level to high level views and the framework also has the potential to support reverse engineering from high level to low level view. It can be semi automatically generated, this semi automation can be reached by using the transformation techniques which has been discussed in section II.

IV. OBSERVATIONS

The framework which has been discussed supports architecture based reengineering and reverse engineering. We have understood the need of UML in software architecture modeling. The various observations that we found while making this framework are categorized as below.

We observed the existing ADL’s always never supports the semantics of the UML while modeling. When we use UML we have to exploit some of the specific architectural rules in deploying the components, configurations and connectors. The artifacts of UML partially support the architecture design.

UML hides most of the information while modeling the problem domain. The domain model must be created separately which is helpful for the creating the architecture with the perception of the architect. Some of the ideas of software architecture completely differ from the UML such as connectors, configuration etc. though UML has the modeling power some of the abstraction do not match the architect’s model. The ADL’s support only a limited number of views (architectural views), the multiple views that has been created using UML makes the process of reverse engineering as a complicated one.

V. CONCLUSION AND FUTURE WORK

We have planned to extend this work in many directions which will be helpful in developing a tool for the UML in architectural modeling, which would be helpful for automation of reengineering and reverse engineering with all the capabilities of the ADL’s and UML’s. The extension of the metamodel can be still extended for a better support for the software architecture. We are planning to generate some additional view which can be added in our framework along with the existing views. We have planned to develop a tool which can integrate all the views of the UML. The tool can be made as an automated one.in future we will still extend the UML diagrams which can provide a high level of abstraction.

REFERENCES
