IJCSC International Journal of Computer Science and Communication Vol. 2, No. 1, January-June 2011, pp. 149-152

# COMPONENT INTERACTIONS FROM SOFTWARE ARCHITECTURE RECOVERY

Shivani Budhkar[1] and Arpita Gopal[2]

[1] Department of MCA, P.E.S. Modern College of Engineering, Pune-411005, India, *E-mail: shivanibudhkar@gmail.com.*

[2] Department of MCA, Sinhgad Institute of Business Administration and Research, Kondhwa, Pune-411048, India, *E-mail: directormca_sibar@sinhgad.edu.*

ABSTRACT

In modern software engineering, Software architecture modeling plays very important role in all phases of software development — like coding, maintenance, testing, etc. Component based software architecture is beneficial as it is useful for reusing system parts represented as components. Most of the existing systems do not have reliable software architecture and some legacy systems are designed without software architecture design phase. So, by doing reverse engineering we can retrieve component based software architecture. The software architecture of the system is described as a collection of components along with the interaction among these components, where as the main system functional block are components, they strongly depend on connectors — which is abstraction capturing nature of these interactions. Therefore, the project will focus on extracting connectors in component based architecture from existing object oriented system.

***Keywords:*** Component Based Architecture, Connectors, Software Architecture, Clustering Algorithms, Object Oriented System.

## 1. INTRODUCTION

Object-oriented development had not provided extensive reuse and computing infrastructures are evolving from mainframe to distributed environments, where object technology has not led to massive development of distributed systems. However, component-based technology is considered to be more suited for distributed system development due to its granularity and reusability. Component Based Development is gaining recognition as the key technology for the construction of high quality, evolvable large software systems in timely and affordable manner.

Using Component based software architecture is beneficial because:

1. Exchange between software architects and programmers easily.
2. Useful for reusing system parts represented as components.
3. Clear separation between components and connectors.
4. Localizing software defects and reducing risk of misplacing new functionalities during maintenance and evolution phases [14].

Software architecture has put forward connectors as first-class entities to express complex relationships between system components [6]. Although components have always been considered fundamental building blocks of software systems, the way the components of the system interact may also be determinant on the system properties. Recent development in the field of software architecture have emphasized the concept of first class connectors which capture the interaction between components and the architectural connectors have emerged as a powerful tool for supporting the description of the interactions. There is a completely new approach to building more reliable software systems which consist to decompose large and complex systems into smaller and well-defined units — software components.

Typically, components are considered to be entities with well-defined provided (server) and required (client) interfaces, and in some cases also with formally specified behavior. A component based application is a collection of individual components, which are interconnected via well-defined connectors between their Interfaces.

Generally, software architectures are composed of components, connectors and configurations, constraints on the arrangement and behavior of components and connectors. The architecture of a software system is a model, or abstraction of that system. Software architecture researchers need extensible, flexible architecture descriptions languages (ADLs) and equally clear and flexible mechanisms to manipulate these core elements at the architecture level [1]. Today software systems are composed from prefabricated heterogeneous components that provide complex functionality and engage in complex interaction. While retrieving component based software architecture, if we identify connectors, these connectors can be reused in similar systems and in more complex

interactions. These connectors are needed to bridge component mismatch or to achieve extra functional properties e.g. security, performance, relia-bility. Also connectors play major role in heterogeneous deployment.

A connector is a reusable entity that models the implements binding among component interfaces. It is inherently distributed; it consists of a number of connector units, with each unit connected to particular component interface [16]. Software connectors perform the transfer of control and data among components. Connectors can also provide services such as persistence, invocation, messaging and transaction that are largely independent of interacting components functionality. Capturing these facilities as connectors helps simplify architecture and keep the architectural focus on domain specific information. Treating these services as connectors rather than component can also help their reuse across domains.

Component contains only the business logic and communicates with one another only via well defined interfaces. The communication paths among the components are in modern component systems realized by software connectors, which allows explicit modeling of communication and also its implementations at runtime.

According to Lubomir Bulej and Tomas Bures [7]. The deployment anomaly problem is inherent to distributed systems and remains one of the strongest motivation factors for introducing connectors as first class entities in component based software architectures. The lifecycle of a component differs significantly from that of a connector.

When using connectors to mediate component interactions, the developers can fully concentrate on application logic, which is often written from scratch, and forget about the implementation details of component interactions, which only need to be written only once and can be reused many times.

The use of connectors allows for greater flexibility when choosing a transport method appropriate for specific interaction.

Current level of work is still insufficient. Although there is some work on the relationships and characteristics of connectors, further step is necessary to construct connectors from existing object oriented system while retrieving component based architecture from object oriented system.

The remainder of this paper is structured as follows. Section 2: Literature Review, Section 3: Introduction to Software Architecture Recovery, Section 4: Methodology, Sections 5 and 6 conclusion and references.

## 2. LITERATURE REVIEW

Various works are proposed in order to extract software architecture from object oriented system. Most of the work uses source code and non- architectural information to extract software architecture. Some of the work has been done using quasi-manual, semi automatic and quasi automatic techniques [14] Most of the components can be extracted but not the connectors.

Here we are trying to extract connectors or connector classes from object oriented system which will be part of extracted component based architecture.

Various works on connectors has been done like classification of connectors [7], interaction mechanisms are constructed compositionally [4], Superposing connectors [6]. Automatic Architectural refinement using Family Design where product design is instantiated using family design [2]. Exogenous connectors are defined which helped in loose coupling in terms of data, function and control [6]. While deploying components connector roles and tasks are defined using connector model which consists of consists of connector frame, connector architecture, connector lifecycle [4]. For software Architecture recoveries from existing system different hierarchical algorithms are compared [14]. By using some set of rules and Object-*z* specification Object Oriented Design can be converted into Component Based Design [14].

Software architectures described using our *C*3 metamodel which is a minimal and complete ADL [1]. A more precise characterization of connectors than in previous work, contradicting the idea that connectors are disjoint from components and discussed, Relationships with coordinators and adaptors [16]. Lubomir Bulej and Tomas Bures proposed an abstract connector generator, a framework allowing to create complex component interconnections almost automatically [7]. Software compositions can be achieved with the help of connector acting as a glue between the communicating components [8].

Work on identification of connectors in software architecture has not been done much.

## 3. INTRODUCTION TO SOFTWARE ARCHITECTURE RECOVERY

### 3.1 Definition of Software Architecture and Architecture Recovery

IEEE defines software architecture as "the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution"; this is closely related to the definition of Shaw, Perry and Garlan [ 5].

Garlan and Shaw [5] define software architecture as comprising components (elements which provide computation services or passive data stores), connectors (elements which provide interactions between the components such as protocols) and configuration (the topology of the system).

Architecture recovery is a part of reverse engineering concerned with identifying architectural components such as subsystems, modules, objects as well as their interrelationships called connectors.

Architecture recovery consists of detection of components and detection of connectors.

Concerning the detection of connectors, most research has been directed towards concurrent and distributed systems as these obviously rely heavily on communication between components.

### 3.2 Elements of Software Architecture

The architecture of a software system is modelled using following design level entities.

- **Components:** Components represent the primary computational elements and data stores of a system. Typical examples of component include such things as clients, servers, filters, objects, blackboards and databases. Components may have multiple interfaces, each interface defining a point of interaction between a component and its environment.
- **Connectors:** Connectors represent interaction among components. They provide the glue for architectural designs. From the run time perspective, connectors mediate the communication and coordination activities among components. Examples include simple forms of interaction, such as pipes, procedure call, and event broadcast. Connectors may also represent complex interactions, such as client-server protocol, or a SQL Link between a database and an application. Connectors have interfaces that define the roles played by the participants in the interaction.
- **Configuration:** A configuration represents graphs of components and connectors. It specifies how components are connected with connectors. This concept is needed to determine if the components are well connected, whether their interfaces match and so on.

## 4. METHODOLOGY

There continues to be great deal of pressure to design and develop information system within a short period of time. So there is need to recover software architectural elements from legacy object oriented system and keep it into repository so that it could be reused as and when needed for fast software development.

Software connectors are important parts of software Architecture which are responsible for interaction between components have an impact on software Architecture recovery [17].

Presently the three important studies on software architecture area are architecture style, architecture connectors and dynamic architecture; it explicitly indicates the importance of connectors [18]. Therefore research on connectors has a very important significance for software Architecture recovery. The objective of this research therefore is to develop an approach for connector retrieval from extracted component based Architecture. Second objective is a variety of connector must be made available for reuse which will be maintained in repository.

This research has significance for several reasons. First it proposes an approach to utilizing connector repository and retrieving the appropriate connectors to enable reuse and help system developers to use corresponding interaction code. Also different views of software connectors are useful for different tasks. In order to model system and communicate its properties detail view is needed.

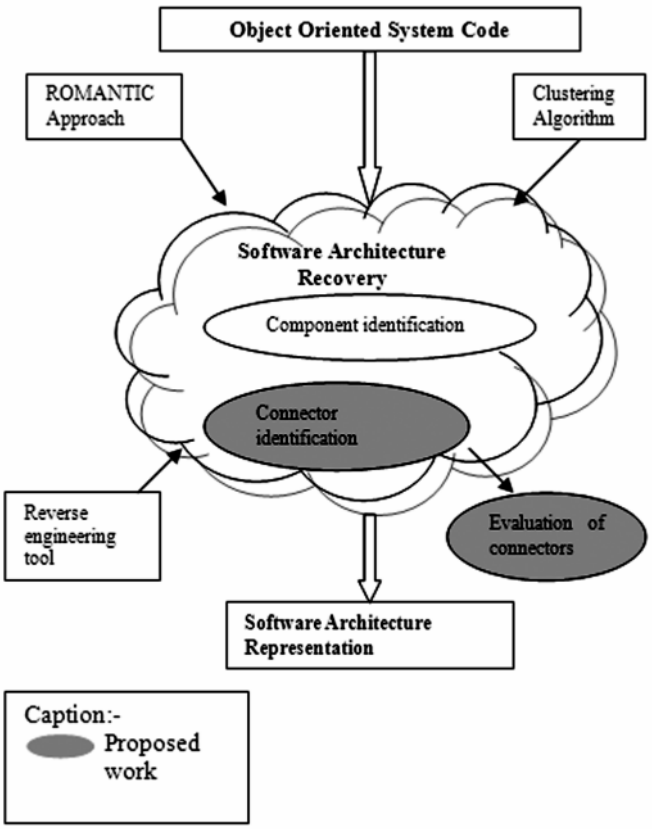
Second it gives systematic construction of connectors as it allows composing simpler interaction into more complex one in an easier way. Also complex interactions can be expressed by nested connector types.

Reuse of connectors is an attractive idea. Current approaches do not capture component interaction effectively. However component interaction has to be reflected throughout whole application lifecycle, otherwise they become serious obstacles in component reusability.

Concretely our goal has been to create a comprehensive approach which would allow us to identify connectors in recovered software architecture and could be stored in connector repository.

Our main work is connector identification from extracted component based software architecture. For this we are going to use existing object oriented system code. Then by using clustering algorithms and architectural semantic constraints component based software is extracted. This contains two parts—component identification and connector identification. First part, i.e. component identification is already done by Sylvain and Mourad [15].

The other method and techniques that will be used in research are ROMANTIC approach proposed by Sylvain and Mourad [2008] using Architectural semantic constraints [15], Study of different clustering algorithms and choose appropriate algorithms, Use of own software reverse engineering tool for retrieving Software Architectural elements, some Software Architecture evaluation algorithm or function can be used.




## 5. CONCLUSION

Connector extraction process uses existing implementation code. While extracting we have to define model, and use clustering algorithms which will map object oriented code elements to architectural elements like connectors. We need to evaluate this architecture is semantically correct.

Once architecture is evaluated as correct one, extracted components are used to define connectors.

We will experiment this process on any particular object oriented system and whatever identified connectors can be put into connector library of that particular domain, which could be reused in another similar system.

## REFERENCES

[1] Abdelkrim Amirat, Mourad Oussalah, [2008]. "Enhanced Connectors to Support Hierarchical Dependencies in Software Architecture", NOTERE 2008, June 23–27, 2008, Lyon, France.

[2] Alexander Egyed Nikunj Mehta Nenad Medivdovic, [2000]. "Software Connectors and Refinement in Family Architectures", *Proceedings of 3rd International Workshop on Development and Evolution of Software Architectures for Product Families* (IWSAPF), Las Palmas de Gran Canaria, Spain, March 2000.

[3] Bridget Spitznagel David Garlan. "A Compositional Approach for Constructing Connectors". *Proceedings of the working IEEE/IFIP Conference on Software Architecture* (WICSA'01).

[4] Dusan Balek, Frantisek Plasil. "Software Connectors and their Role in Component Deployment".

[5] Garlan D. and Shaw M., "An Introduction to Software Architecture", *in Advances in Software Engineering and Knowledge Engineering* (Tortora, V. A. and G., eds.), pp. 1–39, Singapore, World Scientific Publishing Company, 1993.

[6] Kung-Kiu Lau_, Perla Velasco Elizondo, and Zheng Wang, [2005]. "Exogenous Connectors for Software Components".

[7] Lubomir Bulej and Tomas Bures, "A Connector Model Suitable for Automatic Generation of Connectors".

[8] Manzoor Ahmad, Jean Michel Bruel, Antoine Beugnard. "From Composition to Connectors".

[9] Michel Wermelinger, Antonia Lopes, Jose Luize Fiadeiro. "Superposing Connectors", (IWSSD'00).

[10] Nikunj Mehta, Nenad Medvidovic, Sandeep Phadke. "Towards a Taxonomy of Software Connectors". *Proceedings of 22nd International conference on Software engineering,* (ICSE'00).

[11] Onaiza Maqbool and Haroon Bari. "Hierarchical Clustering for Software Architecture Recovery". *IEEE Transaction on Software Engineering*, **33 (11)**, November 2007.

[12] Stephane Ducasse Damien Pollet Löıc Poyet, "A Process-Oriented Software Architecture Reconstruction Taxonomy", *Accepted to CSMR*, 2007.

[13] Stephen Kell, [2007]. "Rethinking Software Connectors SYANCO", 2007, September 3-4, 2007, Dubrovnik, Croatia.

[14] Suk Kyung Shin and Soo Dong Kim, "A Method to Transform Object Oriented Design into Component-Based Design using Object-Z". *The Third ACIS International Conference on Software Engineering Research, Management and Applications*, (SERA'05).

[15] Sylvain and Mourad. "Extraction of Component Based Architecture from Object Oriented System". *Seventh Working IEEE/IFIP Conference on Software Architecture* 2008.

[16] Tomas Bures and Michal Malohlava and Petr Hnetynka. "Using DSL (Domain Specific Language) for Automatic Generation of Software Connectors". *Seventh International Conference on Composition – Based Software Systems*, 2008.

[17] Trevor Parsons, Adrian Mos, Mircea Trofin, Thomas Gschwind. "Extracting Interactions in Component Based Systems". *IEEE Transaction on Software Engineering*, June 2008.

[18] Zhang Jingjun, Li Hui and Li Furong. "Research on Aspect Connectors for Software Architecture Adaptation".