Extendable and Interchangeable Architecture Description of Distributed Systems Using UML and XML

Chang-ai Sun[1,2], Jiannong Cao[1], Maozhong Jin[2], Chao Liu[2], Michael R. Lyu[3]

[1] Department of Computing
Hong Kong Polytechnic University
Hung Hom, KLW Hong Kong
{cscasun,csjcao}@comp.polyu.edu.hk
http://www.comp.polyu.edu.hk/people/csjcao.html
[2] School of Computer Science and Engineering
Beijing University of Aeronautics and Astronautics
Xueyuan Road 37, Haidian district
100083 Beijing, P.R. China
{jmz,liuchao}@buaa.edu.cn
[3]Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin N.T. Hong Kong
lyu@cse.cuhk.edu.hk

Abstract. Software Architecture can help people to better understand the gross structure and, with powerful analysis techniques, to evaluate the properties of a software system. To accommodate the dynamic changes and facilitate interoperation of tools, an architectural description of the distributed system should be extensible and interchangeable. In this paper, we utilize the built-in extension mechanism of the Unified Modeling Language (UML) to describe the architectures of distributed systems, with the underlying architectural metadata represented in XML. In particular, the approach has been applied to describe the architectural model of distributed software in the Graph-Oriented Programming framework. The proposed approach has many desirable features, characterized by being visual, easily extendable and interchangeable, and well supported by tools.

1 Introduction

Software Architecture (SA), as a bridge between requirements and design [1,2], is a high level abstraction of system structure. It is composed of a set of components with independent functions and explicit interfaces and interactions between the components. Software architecture description is the representation of abstract architectural model in Architecture Description Language (ADL). It provides a blueprint for system construction and composition, and permits designers to make early design decisions based on the architectural documentation. Various ADLs have been proposed in the community of SA, such as Darwin, C2, Rapide, Aesop, Wright, SADL,Acme[3].

Software architecture description can be characterized by various features, including composition, abstraction, reusability, configuration, heterogeneity, and analysis [6]. Furthermore, it has been argued that software architecture description of distributed systems should be dynamic and reflective [7]. With the development of many well-known ADLs, some additional issues need to be addressed. For example, how to decide the tradeoff between visualization and rigorousness is still a key issue to design ADL. Another problem is how to exchange description between the architectural models described by different ADLs. It is our opinion that architecture description of a distributed system should also be:

Extendable: The architecture description should be able to be extended as required. For example, the support of time performance analysis is a key requirement of the architecture model of Real-time systems, while it may be neglected in early systems.

Interchangeable: The architectural representation should be able to be exchanged between heterogeneous environments, including different languages and platforms. This property is particularly important for distributed systems.

There are some attempts to address the interoperability, including: (1) The development of an architectural interchange language, e.g. Acme [4]. (2) The establishment of methods to integrate architecture-based tools [2]. (3) The definition of ADL based on XML schema, such as xADL [5]. The first approach requires that ADLs involved in architecture description interchange be homogeneous. This is hard to imagine because any individual ADL is designed for special types of applications or features. The second approach leaves the task of translation to the tools, which is impossible in some cases. The third approach utilizes XML, which makes the description easily extensive, and more the physical model easily exchanged. For this approach to be successful, however special graphic notations for architectural elements must be redesigned for the feature of visualization, and thereby some corresponding supporting tools are also developed.

This paper proposes an approach to develop description of architecture of distributed systems, which is extendable, interchangeable and well supported by standard graphic notation. The core idea of the approach is to incorporate UML and XML. In particular, rather than translating an ADL to UML, we will extend UML based on its built-in extension mechanism, to directly describe an architectural model for distributed systems, namely the Graph-Oriented Programming (GOP) Model [7].

The rest of this paper is organized as follows. The next section describes the proposed framework of describing the GOP architecture. Section 3 proposes an approach to extending UML to describe the GOP architectural model based on UML extension mechanism and XML and provides an example of GOP demonstrating how the proposed approach can be used. Section 4 concludes this paper by offering some conclusions and plans for future works.

2 The Framework of Describing the GOP Architecture

In this section, we introduce the underlying concepts of GOP model and present the framework of describing the GOP architecture.

2.1 The GOP Model

The Graph-Oriented Programming Model (GOP Model) was originally proposed for distributed programming [7]. Under the GOP model, the components of a distributed program are configured as a logical graph and implemented using a set of operations defined over the graph. In GOP, a distributed program is defined as a collection of *local programs* (LPs) that may execute on several processors. Each LP performs operations on available data at various points in the program and communicates with other LPs. The GOP model is shown as Fig.1:

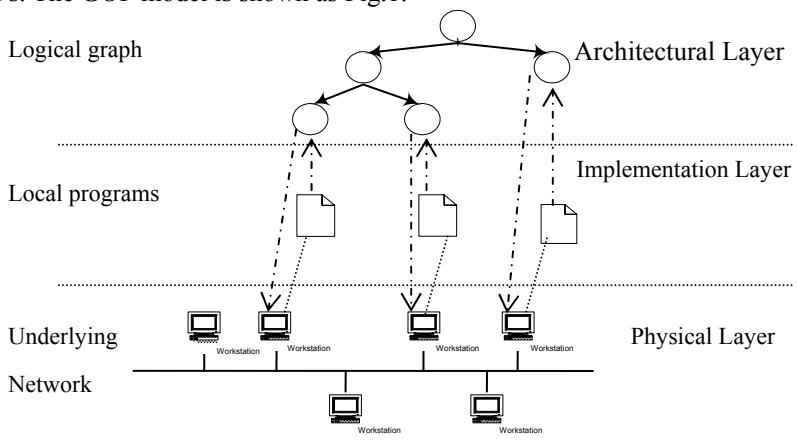

Fig.1. The GOP Conceptual Model

As illustrated in Figure 1, the GOP model consists of:

- a ***logical graph*** (directed or undirected), whose nodes are associated with local programs (LPs), and whose edges define the relationships amongst the LPs.
- a set of ***local programs*** (LPs) performing the computation tasks and cooperating with each other using the graph-oriented programming primitives.
- a ***LPs-to-nodes mapping***, which allows the programmer to bind LPs to specific nodes,
- an optional ***nodes-to-processors mapping***: which allows the programmer to explicitly specify the mapping of the logical graph to the underlying network of processors.
- a library of language-level graph-oriented **programming primitives**.

It is noted that the logical graph in Fig.1 is very similar to the architecture of distributed program, according to the definition of Software Architecture discussed in Section 1. Further, this *architecture layer* of GOP Model relates to the *implementation layer,* which consists of a set of *Local Programs*, and the *physical Layer,* which consists of a set of processors connected by the underlying network.

When the distributed programs following the GOP Model are implemented, the Logical Graph is used as a type object, implementing the expected functions together with all the local programs. That is, the graph-oriented architecture model has been incorporated into the runtime system. Therefore, the architecture of distributed pro-

gram can be dynamically *reconfigured* with *explicit reflection*, by invoking the behaviors provided by a logical graph object. It is a highly desirable feature in building dynamic distributed system.

We have leveraged the GOP Model as the architectural model. All processors will need to share the architecture representation of GOP Model, so the architecture description should be easily interchangeable between the different processors. It is especially true, as a result, for the heterogeneous distributed systems. Next, we will focus on extending UML to describe the GOP-based architecture.

2.2 The Framework

The principle of extending UML to describe the architecture model of GOP is illustrated in Figure 2. The central idea is to describe the architectural level information and semantics (including concepts and properties) of the GOP Model, a kind of architecture model, based on UML infrastructure (standard modeling notations and built-in extension mechanism) and XML.

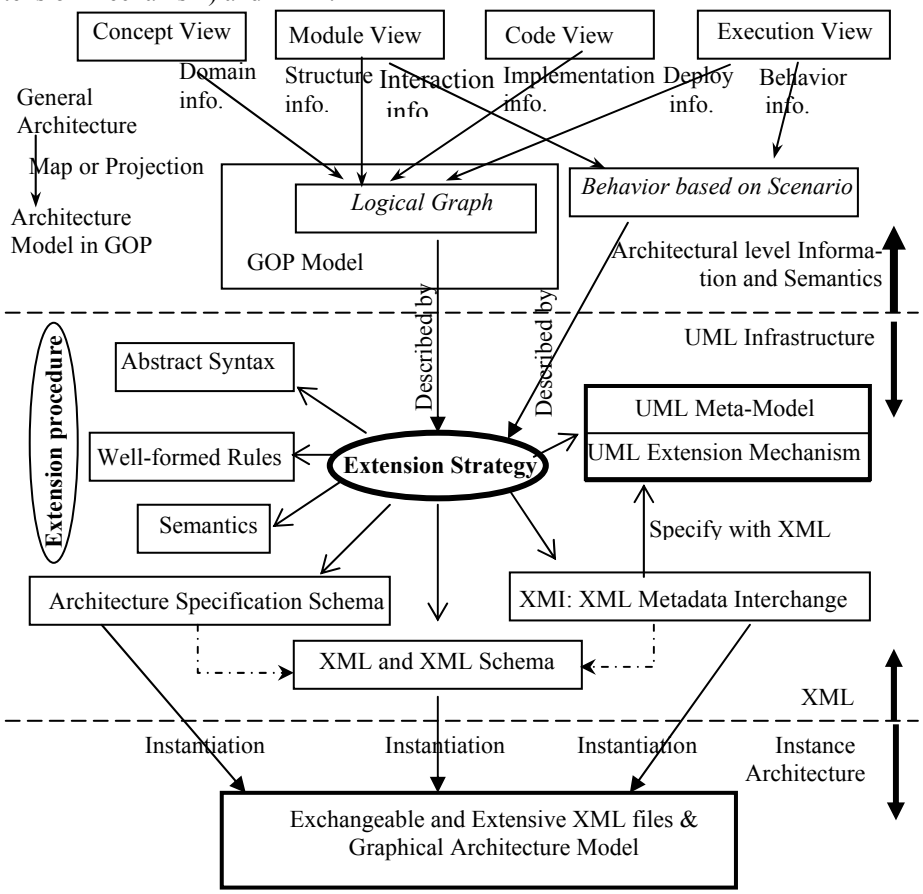

Fig.2. The Framework for Describing GOP Architecture Model with Extensive UML and XML

The framework of our approach is divided into three layers. The top layer is *Architectural information and semantics*, which is concerned with what should be described for software architecture. Four common recognized architecture views are mapped or projected into the *logical graph* of the GOP model. The middle layer is *UML infrastructure and XML*. *Extension strategy*, the key element of this layer is concerned with issues, such as which Meta-model of UML is selected and extended to describe logical graph and behavior based on Scenario, how to extend them, and shat are the steps to follow for extension. The bottom layer is *Instance Architecture.* An architecture instance can be described as a group of graphical architecture views or a set of interchangeable and extendable XML files.

3 Extending UML to Describe GOP Model

In this section, we propose an approach to extending UML to describe the architecture model in GOP.

3.1 UML Built-in Extension Mechanisms and XMI

UML [8] is a standard object notation for industry. A UML Model of a software system generally consists of several partial models, each of which addresses a certain set of issues from different angels and at different levels. UML is graphical language with a fairly well defined syntax and semantics. The syntax and semantics of the underlying model are specified semi-formally via descriptive text and constraints, such as OCL expression. The linguistics architecture of UML is a four-layer meta-model, and the *meta-model layer* is a focus for *extension* and the *development of tools*. It is noted that the *meta-model* of UML itself is organized in the form of package.

UML is an extendable language in that new constructs may be added to address new issues in software development. Three mechanisms are provided to allow extensions without changing the existing syntax or semantics of the language: 1) *Stereotype* allows groups of constraints and tagged values to be given descriptive names and applied to other model elements, 2) *Tagged Value* allows new attributes to be added to model elements, 3) *Constraint* allows new semantics to be specified linguistically for a model element.

To enable model representations generated by the different supporting tools to be exchanged between the tools, XML Metadata Interchange (XMI) Specification is also provided in version 1.4, and later version.

3.2 Extension Strategy

As discussed in Section 3.1, UML provides a set of extension mechanism to satisfy new modeling requirements. Some standard stereotypes are also provided in the meta-model of UML for general extension. The process of extending UML to describe the architecture can be summarized as follows:

1) Select an appropriate meta-model package to extend. The meta-model selected should be the one closest to the architectural view from the perspective of semantics.

2) Decide the Meta-Class to use as the base-class of standard element in the UML meta-model.

3) Associate architectural semantics to the elements of the architectural view, including some extended attributes or specific constraints. Here, extensions for different architectural views and architecture properties can differ greatly.

The architectural model using UML can be described with different extension strategies. David Garlan et al. proposed that different extension strategies for architecture structure should be evaluated by using three evaluation criteria: *Semantics-Match, Legibility,* and *Completeness* [9]. Extension strategies should also be: 1) *Extensible and well-structured*, meaning that architectural description should be able to be enhanced further to support the description and analysis of new properties. 2) *Efficient and Effective*, meaning that the extension should as far as possible reuse the features of UML supporting tools to serve the requirements of the architectural description.

3.3 A Reference Extension for the Logical Graph

As an illustration, we propose a reference extension for the logical graph. First, we must choose the meta-model package closest to the logical graph in semantics. Comparing UML Component and UML Class, it's more suitable to select the meta-model of Class to extend for describing the logical graph, based on the extension criteria discussed in Section 3.2 and the following observations:

1) UML Component focuses on physical implementation, while logical graph focuses on logical structure.
2) As the core element of OO, UML Class can specify systems from several abstract levels, such as concept, specification and implementation.
3) The relationships between UML Classes are also richer than those between UML Components. This can make it easier to depict the interaction of components.

Once we decide to choose UML Class to extend for describing the node of logical graph of GOP, the next thing is to decide the Extension Procedure. In order to maintain consistency with the UML manual, we take the following steps: *Abstract Syntax, Well-formed Rules, Semantics* and *specification*A reference extension to the Core Package of Foundation package of UML meta-model is illustrated as Figure 3.

Step 1: Abstract Syntax

In Figure 3, <<LGcomponent>> is used to describe the components of the architectural model from three perspectives, namely *context, specification*, and *implementation*. We'll focus on the specification of a component, which consists of:

Name: is inherited from ModelElement::Class

isOption: indicates whether the component takes part in the current configuration

isComposite: indicates whether the component is a composite component

Type: abstracts the function of the component described in certain kind of specification language

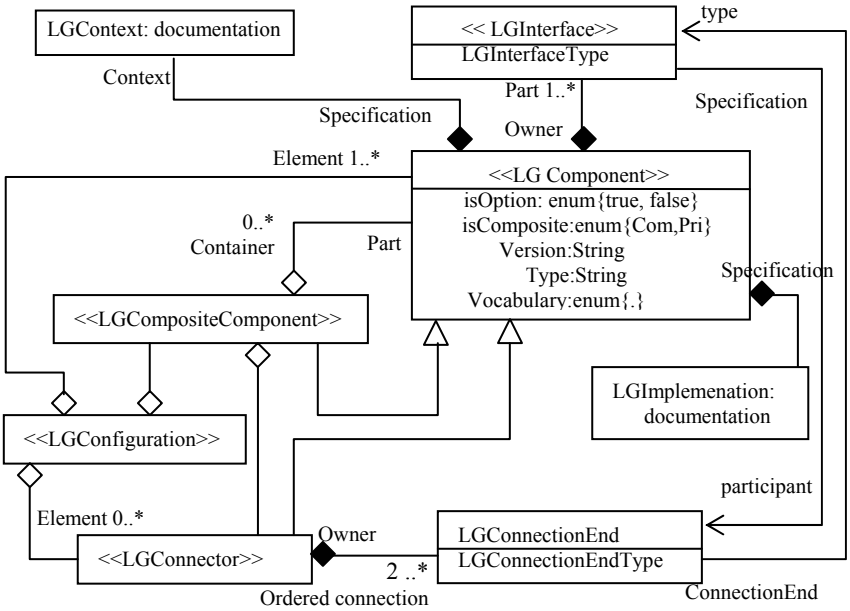

Fig.3. Meta Model for the Logical Graph of GOP

Version: is used to identify the version information of the component

Vocabulary: indicates special type or customary component, such as Filter or Pipe. Preserved for Architecture style or Pattern.

LGComponent assembles Interface, Context and Implementation, by means of the association.

Implementation: the extension used to indicate the implementation of components. It can be one or more artifacts, such as binary, executable, or script files.

Context: the extension specifying the scenario in which the component can operate as expected.

Interface: the extension that names the points through which the component can interact with the environments, including asking for and providing services.

CompositeComponent: the extension used to indicate the compositecomponent in which the component takes part.

LGConnector, LGConfiguration and LGCompositeComponent can be specified in the similar ways. But they will not be further discussed here.

Step 2:Well-Formed Rules

Some well-formed rules must be followed when using the extended UML model to describe the logical graph of the GOP model. Otherwise, the extension will not be interpreted appropriately. Once again, for brevity, we will demonstrate only a few important well-formed rules.

LGInterface is an instance of a meta-class Interface. It defines the interface of LGComponent and must satisfy following constraints:

1) The contents of LGInterface must be public or protected operation (Interfaces may not have Attributes, Associations, or Methods).

Self.allFeatures->forAll(f| f.visibility=# Public or f.visibility=# Protected)

2) Only two interface types are allowed in the LGComponent.

LGInterfaceType: Enum{Request, Service}

LGComponent is an instance of meta-class Class. It incorporates many tagged values for describing architecture attributes. Some enhanced constraints must be satisfied in our architectural model, including:

1) A LGComponent must have at least one Service Interface.

self.allInterfaces->exist(i| i.LGInterfaceType=#Service)

2) Any attribute of a LGComponent can be accessed only by the LGInterface.

Self.OclType.feature->forAll(f| f.oclIsKindOf(Operation))

Step3: Semantics

LGComponent is a basic construct of the logical graph. A LGComponent consists of at least an interface to communicate with external environments. Communication with LGComponent can only occur through the LGInterface. LGComponent can be categoried as Atomic and Composite component, the latter is composed by many LGComponents and LGConnectors. A composite component can also be a part of one or more larger composite components.

Step4: Specification

The results of a graphical architectural description must be translated into the textual specification or other binary files. To be exchanged between different UML tools and other architecture supporting tools, the specification should be represented in XML based on the base of XMI. Since the extended meta-model is an architecture type rather than an instance architecture, we use XML Schema to represent extended meta-model. To effectively represent extended meta-model, the XML schemas must be well structured, we organize XML schema into a hierarchy structure [1].

3.4 Case Study: a Simple Example

In this section, we will illustrate our approach using the calcpi program used in [10]. This program computes an approximation to π by calculating the area under the curve 4/(1+x2) between 0 and 1 using numerical integration. The program is structured as a set of *workers,* each computing the area, and a *supervisor* collecting the results and averaging them.

We will represent *worker* and *supervisor* as *components*, which execute different calculations, while the communication between *worker* and *supervisor* is treated as *connectors*. In the following example, there are two workers. The *logical graph* representing the architecture of this distributed application is described by extended UML meta-model as shown in Figure 4.

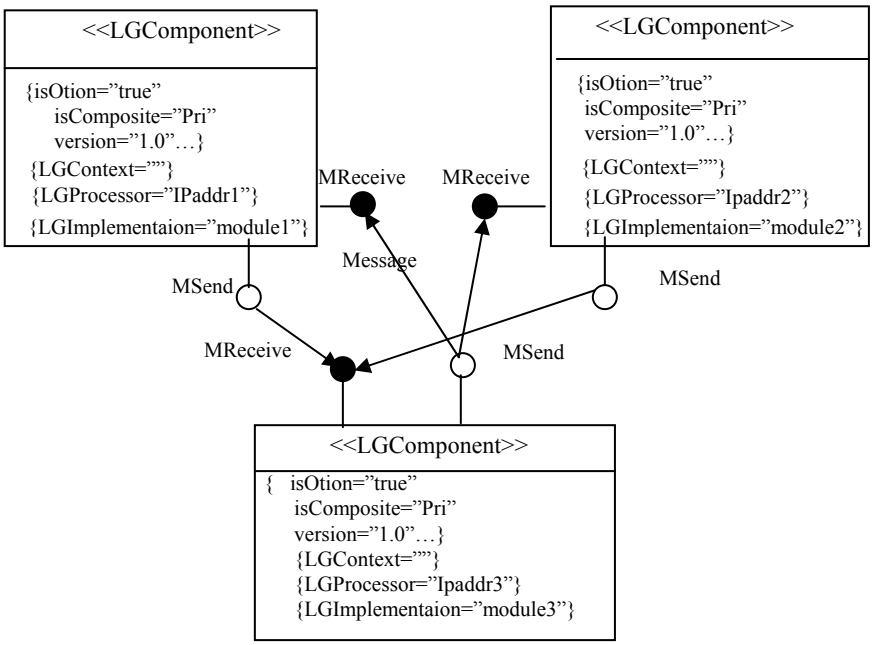

Fig.4. The Architecture Model Described by Extended UML Class Diagram

The extended UML Sequence Diagram can be used to describe collaborations between *worker* and *supervisor*. In the GOP Model, *nodes-to-processors* mappings are used to deploy the logical graph on the physical networks environments. This can be implemented by specifying the tagged value LGComponent, namely LGProcessor. Similarly, when all components are assembled to run, *local programs* must be bound to the *worker* and *supervisor* of logical graph. This is implemented by *LPs-to-Nodes mappings* in GOP model. In our approach, this can be implemented just by specifying LGImplementation of LGComponent. Moreover, *Version* of LGComponent can be labeled with a different version number to recognize the changes of the implementations of components. The specification for architectural configuration and components of this distributed application described in XML is omitted for the limitation in space.

4 Conclusions

This paper has proposed an extendable and interchangeable architecture description approach based on UML and XML. The idea of our extension strategy can be summarized as follows. 1) UML class is extended as component, 2) UML interface as the interface of component, 3) UML association as connector, 4) UML subsystem as architecture configuration.

Compared with the existing work [11,12], our approach can be characterized by the following features: 1) *Visual modeling* of architecture of distributed system by reusing one standard modeling language UML and its supporting tools.2) Resulting architecture specification can be *interchangeable* or shared by different supporting tools. 3) *Flexible extension* can be further obtained. Here, we examine some derived features of our approach, and explain why or how they can be obtained.

Integrated Version Management in architecture level can be obtained. In our approach, the component and configuration can be labeled version. If we want to modify a component of the distributed system, the version number of component has been changed from the perspective of version management. This is also easily implemented by modifying the version attribute of node in *Logical Graph.* At any time, the configuration of logical graph maintains the version profile of the current system. When the system evolves, it can be characterized by the evolution of configuration version. If the version information is not required, just set all the version attributes of component and configuration to default or neglect them.

Visual Architectural Description can be supported. UML notations can be reused by our approach, but the semantics for notation of UML is changed, which is entitled with constraints and architectural semantics. For example, LGComponent can be represented as the notation for UML Class, but the semantics must be interpreted as semantics of LGComponnent discussed in Section 3.

Existing tools for UML can be reused. Our extension approach is compatible with UML manual, and XMI is accepted as the base of architectural specification, so most of functions provided by UML supporting tools can be reused to describe software architecture.

Further Extension for new architecture properties can be easily obtained. Here, we give an illustration of the further extension. In GOP model, the logical graph can also be deployed on the physical environments while the system is running. This can be implemented by adding an attribute *LGProcessor* to the construct LGComponent, an item responsible for this attribute should at the same time be added into their architecture specification.

Our future work will be conducted to integrate further our approach and the GOP model, including the development of supporting environments, mapping the specification into different platforms, such as Web, Component and Cluster, and representing some large-scale real world applications using our approach in this paper.

Acknowledgement

This research is partially supported by the Hong Kong Polytechnic University under the research grant H-ZJ80, the National High Technology Development 863 program of China (No. 2001AA110244), and RGC Project (No.CUHK4360/02E).

References

[1]Chang-ai Sun. Contributions to Software Architectural Description and Construction and Reconstruction.[PhD Thesis]. Beijing University of Aeronautics and Astronautics, 2002.12

[2]David Garlan. Software architecture: a roadmap. In proceedings of the conference on the future of Software Engineering, (2000) 91–101

[3] Nenad Medvidovic, Richard N.Taylor. A Classification and Comparison Framework for Software Architecture Description Languages, IEEE Transaction on Software Engineering, Vol.26 No.1, January 2000,70-93

[4] D.Garlan, R.T. Monroe, D. Wile. Acme: An architecture description interchange language. In proceedings of CASCON'97 Ontario, Canada, November, (1997) 169-183

[5] Eric M. Dashofy, André van der Hoek, and Richard N. Taylor A Highly-Extensible, XML-Based Architecture Description Language.In Proceedings of the Working IEEE/IFIP Conference on Software Architectures (WICSA 2001), Amsterdam, Netherlands.

[6]Mary Shaw, David Garlan. Characteristics of higher-level Languages for Software Architecture, Carnegie Mellon University, Technical Report, CMU-CS-94-210, 1994.

[7]Jiannong Cao, Xiaoxing Ma, Alvin. T.S. Chan, and Jian Lu, Architecting and Implementing Web-based Distributed Applications Using the Graph-Oriented Approach, to appear in Software: Practice and Experiences (John Wiley & Sons).

[8]OMG, Unified Modeling Language Specification (Ver 1.5), Mar 2003

[9] D. Garlan, A. J. Kompanek and P. Pinto. Reconciling the needs of architectural description with object-modeling notations. In proceedings of the Third International Conference on the Unified Modeling Language, York, UK, October 2000.

[10] N.Rodriguez, R. Ierusalimschy, and R. Cerqueira, Dynamic Configuration with CORBA *Components,* In proceedings of the Fourth International Conference on Configurable Distributed Systems, IEEE Computer Society Press, May 1998, 27-34

[11] Nenad Medvidovic, David S.Rosenblum , David F.Redmiles, Jason E. Robbing. Modeling software architectures in the Unified Modeling Language, ACM Transactions on Software engineering and Methodology, Vol.11, No.1, January 2002, 2 - 57

[12] C. Hofmeister, R. L. Nord and D. Soni. Describing software architecture with UML. In Proceedings of the First Working IFIP Conference on Software Architecture, San Antonio, TX, February1999, 145-160