Abstract

Multimedia communication aims at transferring various forms of data, audio and video simultaneously in a single network. The applications such as videoconferencing, digital TV, multimedia teaching, distance learning and video on-demand make use of multimedia network and communication technologies to provide quality of service (QoS). With the wide variety of different real time system applications we need specific software for those specific real time systems that will meet the requirements of an application.

In this paper, we propose the design and development procedures for real time systems using component based methodology. It starts with the requirement analysis of a real time system and continues to provide the functional model of the system and also the environmental design as well. As more and more functions or components needed to be added in designing a software system a modularized and systematic approach becomes not an option. Hence this paper proposes a systematic and procedural approach that will help to build a more reliable and suitable software architecture. This paper also includes a case study – a Multimedia Real time conferencing system, whose aim is to provide a brief demonstration of how to apply the architecture and notations that are presented in this paper.

Keywords: Real time systems, Software architecture, Object-Oriented modularization, Component discovery and notational representation, Architectural design.

1. Introduction

The objective of explaining what real time systems are is followed by presenting some of the most common definitions of software architecture. To start with real time systems are those which need the results within definite timescales, otherwise the system just won’t work properly.

By the word timescales we mean the system is bound by specific deadlines for the task to be finished. The basic component of a real time system consists of a controlling system and a controlled system. A controlled system can be any real time environment that has to be monitored by computer software. This paper describes ‘Real time’ as those systems with time-bounded response constraints by which we mean if the computer system response exceeds the specific time bounds then performance degradation or malfunctioning will result.

The aim of the work in this paper is to present a new radical approach for building software architecture for real time system. Defining software architecture would help us understand the future literature of this paper.

Definition Software architecture is the high-level structure of a software system or the organizational structure of a system. Architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts [1].
Every software architecture should follow a specific methodology. There are many different software methodologies. We use Object-Oriented methodology to design software architecture.

2. Related works

Formulating a general methodology to the development of real time systems must meet the high performance, size constraints and functionality. Many methodologies have been proposed before and most of them have focused on representations that used graphical notations, particularly flow charts.

There are many variations that became more prevalent in 1970’s. One of the phases of RTSAD had a behavioral model and environmental model. As the name suggests, the environmental model defines the environment in which the system has to perform and behavioral model defines the behavior of the system.

With the growth of structured programming in the early 1970’s, the idea of top down design and stepwise refinement gained prominence as program design methods, with the goal of providing a systematic approach for structured program design[4].

In 1970’s Dijkstra’s hierarchical design method aimed at dissemination the entire system into layers or several layers and modules. Modules in each level provided a service to a higher lever module. This was the first design method to address the concurrency in a real time system. In order to inculcate a more structured design, a methodology which represented the system into functions and interfaces in communication between each module became more prominence. To represent the behavioral characteristics finite state machines were used. Module cohesion and module coupling are used to evaluate the methodology.

An alternative approach was also designed in which emphasis is on the first designing program structures based on the data structures. Jackson system development is an instance of the above methodology. In there, the design is based on modeling the reality at first before any function of the system is being considered. This approach is much like a simulation and each is modeled by means of a concurrent task called a model task. This design methodology used entity structure diagrams rather than individual symbols.

But many still had difficulties in finding a design methodology for those programs using global data structures. The information hiding technique largely solved this problem. The NRL methodology applied the information hiding concept for large scale systems. In this each module that is considered to be changed is hidden in a separate information hiding module. This design methodology is a major contribution to software design.

Object-oriented design used abstraction to separate object specification form its body. Information hiding is a major aspect of this method. This paper is about another object-oriented methodology that will help software designers to build efficient real time system.

3. Functional model of a real time system

Building software architecture for a real time system needs a precise understanding of what exactly a proposed system should do. This is the fundamental purpose of the analysis and specification – the requirements phase.

3.1. Functional model of a real time system

Specification requirements are exactly where our framework for the architecture starts. We first aim at identifying the key functions of any real time system. These functions are then
modeled and defined precisely with different language tools. We identify some of the most common and widely known requirements of a real-time system as the first step.

They are as follows:

- The system must provide a way to sense the real-world changes and convert them into information.
- The system must have hard-fast/soft-slow design
- The system should give the user the required information not after the deadline
  - If hard-fast, then exactly before the deadline
  - If slow-soft, the system may be time-bound but still should not take more than some few seconds after the deadline.
- The system should process and validate the information sent by the external devices.
- A good graphical, user-friendly interface should be provided.
- The system must have a repository to store the processed data.
- It must provide a robust means of controlling the flow of data.

3.2. Functional design

The functional view of a real-time system helps us identify precisely what each function or module is supposed to do once they are integrated within the system. Once these functions are clear, then building a more complex and complete real-time system would not be a difficult task. Figure 1 shows the pictorial representation of the functions a real-time system will perform under a given condition.

The function of the first block is to collect the data from the real world. This data is then processed and should be validated. The validation is a more significant aspect of real-time system as these processed data are those which are used to control back the objects in real-time systems. Once the data is validated, it is then stored in a repository for future retrieval or query process. This explanation briefs only the basic functions that a system must perform to be called a real-time system. One can always branch out and find more specific functions depending on for what a real-time system is used. Also, identifying and defining each and every possible function for a real-time system is not feasible and is out of the scope of this paper. Given a real-time system formulating its functions is up to the audience of this paper.

![Figure 1. Functional view of real time system](image-url)
4. Architecture design

Figure 2 explains development environment to help the software designers design the better and more efficient software architecture for real time systems. This environment gives to the user a graphical view of the architecture built using the basic notations summarized later in this paper. Since most of the real time systems communicate with the objects in the outer world an object-oriented approach suits well with the world which deals only with objects.

Also objects have its communication patterns and behavioral patterns.

**Figure 2. Structure for real time system**

4.1. Specification classifier

The environment for real time software architecture is best understood with the design. As said earlier, formulating a requirement specification for the system would be a key aspect to begin with. Development environment then classifies the requirements specification into embedded system specification and software specification. This helps designers to simplify the job of identifying components and its functions. Hence we highly values object-oriented approach at all levels. All the real time system would have several forms of embedded system and the system will vary depending on the tasks which system is being built. For instance, a smoke detector is a simple real time system containing embedded system with one micro controller. On the other hand a large real time knowledge system should have more efficient multi processors with memories at least double as compared to the conventional real time systems.

4.2. Specification analyzer

The embedded system specification analyzer module evaluates the modeled architecture to check whether all the specification requirements have been met. A repository module stores the modeled system for the future use. Though there are no notations yet, we will use the basic notations to design the architectural details of the system. Every device is represented as object and each object is given its own attributes and behaviors. Also associations between each object are represented using the notations given in this paper. The object in the system communicates using specific signal types.
4.3. Architectural semantics and syntactic elements

A notational diagram and its semantics are stored in a file for the designers to use. These descriptions have both the semantic and syntactic details for the system. For example, one may use function names like call() that will be used as a procedural call. Its syntactic details consists of how and where a call should be used and in what format while its semantic details consists of what the function does i.e., calling another function or subroutine.

4.4. Elements of software architecture

Before the software architecture notations are described it is essential to explain some of the most significant aspects of the software architecture. Some of the most common elements to be explained are Packages, functionality, quality requirements and architectural styles [10]. The explanation for the software architecture elements presented in this paper is the form of various research groups and research papers.

Every notation is represented by both components and connectors. The components are the real world entities in the system or a device in a system. The connectors are the interfaces between two components and they define the relationship between two components.

The components which have same property or the components which behave in a similar way are grouped into same modules. Those grouped components form packages. Each notation has its own representation of packages and components. We use the basic concepts of object-oriented approach to represent a component.

Functionality is another important element of software architecture. The functionality of a system represents the behavior of a system at a specific time. This is usually implemented at the programming level.

Quality requirement is another aspect which deals with the robustness, security, scalability and other performance aspects.

Finally architectural styles give the designers to choose between different styles available based on the type of data flow. The following examples are from [7].

These components are used in the design of the software architecture. The environment presented in this paper provides the embedded system architecture suitable for the specified application and the system software architecture as well.

![Figure 3. Notations in our systems](image)

5. Frameworks for the system model
In this framework each module processes an event and the result of each event is sent to the appropriate components. In this section we describe object orientation followed by the software model designed using object-orientation. The architecture design diagram for real time systems is shown in figure 4.

This architectural design uses object-oriented approach. Also we represent each object with its own notation so that the designers don’t be confused when they use too many notations. Though the design seems to resemble layered approach the object-oriented concepts has been largely used.

5.1. Procedure and guidelines to develop the architecture

A good understanding of the system specification and analysis of that specification is the key activity in requirements analysis phase. In conceptual design phase a high-level design of the system is built. In this phase the system design would not contain the detailed architecture but rather an overall structure. This is an abstraction level design where most significant object and their behaviors are identified ignoring other unimportant objects.

Once we have the conceptual design we now have to define the functions of each object in the real time system. In this phase a pictorial representation of the functions or tasks carried out at each stage of the system is provided. This functional view will be of great use for the designers when they reach the implementation phase of the design. Before we start designing a blueprint for a system we need to identify what objects or external devices should be used based on the functional view developed in the previous phase. This is done in the object discovery process where we highlight all the nouns or noun phrases in the problem domain.

In this process the objects of interest are first identified and then the object of less interest are highlighted. Once we find the objects of interest we need to classify them into active and passive objects. Active objects are those request services from a passive object. Passive objects can be considered as servers.

This phase also includes identifying the physical devices that should be used in a real time system. Also identifying the key concept for using an object at this stage will largely reduce the problem in the future.

A descriptive model of the high-level design we provided before is built in this phase. This descriptive model expands the high-level design into a more detailed description of the system. This descriptive model helps to identify functions and partitioning of subsystems, type of interfaces between objects, application of specific solution to sub problems etc. Modularization is one of the important aspects in object-oriented approach where one large problem is being broken down to many sub problems. We can reduce the complexity of the problem largely in this way. This is achieved by first identifying the boundaries of each objects and how they interact with other objects.

Once we are clear about these details then we can break down a big module into chunks of smaller module. Each problem is then handled by different person. This is the final phase of the guidelines for building real time system. There would be many different kinds of testing. Correctness, reliability and maintainability of the system should be tested. Also each individual unit or module should be tested for the interoperability and scalability issues. But we will not discuss about them in this paper.

5.2. Software model and components

All real time systems use some kinds of external devices to communicate to the real world. These are called sensors and actuators. These objects are represented using nodes, the basic
component defined before. Each node should be able to explain the attributes and its behavior at a particular state. Each component given in the Figure 4 contains a function that performs a specific action. Once these functions are executed with no errors components produce a set of intermediate result or final result.

One of component’s output may be an input for another component. In such situation the component which produced a result is referred to as producer agent and later is referred to as successor agent.

These can also act as either a passive object or active object. In this framework a State manager object maintains the status of a passive and an active object. This helps to maintain a record of which component is currently producing result and which is consuming. Also with the use of state manager object we can maintain a record of each object’s status and function description. These records are then stored in the repository operated by Database Management module. Any alteration for a passive object needs only one change at the central database. Hence redundancy is not an element to be questioned.

Each basic building of the system is connected using other objects called connectors. These connectors are responsible for the data transmission between components and devices. Each time a data is transmitted from one component to another the component which sent the data is represented as a –ve component and the component which received is represented as +ve component. This information is attached along with the data object being sent. Each time a data is being received it is checked for the validity by Data Analysis module.

5.3. Embedded system architecture

This is an important module in real time system and is carefully designed depending on many constraints and specifications defined earlier. The environment development unit clearly
separates and identifies the embedded system specification from the software specification with the intention of making designers' job simple. We all know that an embedded system should contain one or more microcontrollers to perform some action. This architectural design assumes that most of the real time systems that are built now need more than one microcontroller as today's necessity has increased largely. These microcontrollers are again represented as individual objects. This diagram is a functional representation of the real time system. The data flow in a real time system is represented using notational connectors.

6. Case study

The main objective of this case study is to apply or follow the procedures to build a new software architecture from the architecture provided before. This way the validity of the software for real time system is checked.

The distributed multimedia real time conferencing system is a more advanced technology that allows geographically dispersed users to concurrently create, edit, and discuss shared multimedia documents. Distributed system supports the digital audio and video with network and the operating system with continuous media flows. These were streams of data generated by the applications that produced data objects periodically.

The model is configured to cover diverse geographical distributions such as LANs interconnected through MANs and WANs or can be configured to allow inter-LAN conferencing. First for any real time system the functional view needs to be defined.

All the objects are defined in the component definition stage. This case study will follow the exact procedures and guidelines as shown in Figure 5.

![Figure 5. Guidelines to develop the architecture](image)

6.1. Requirement analysis

The first step in the flow chart for building software architecture is clearly defining the requirements of the system. Multimedia real time conferencing system should:

- facilitate the users to distribute any multimedia related applications or files.
- be possible for geographically dispersed users to concurrently create, edit, and discuss shared multimedia documents.
- have a good concurrency locking system.
- be up and running all the time.
- support different types of computer system.
6.2. Component identification and definition

Multimedia real time systems is a client/server based architecture with 3 main components as followings:

- Directory servers
- Conference Managers
- Conference Agents

These are a collection of conferencing system object. Directory servers provide user authentication and name-to-address mapping functions. Each directory server is responsible for a domain, or position of a network, which consists of one or more conference managers. So these conferencing managers can be grouped into a package of directory servers. These directory servers can be altered to add or delete domain name or addresses in the directory servers.

These directory servers can be represented as a top level directory servers or it can be put at any level as a top level for other conference managers. So with our notations these directory servers can be represented either as a node component or as a module component. These object-oriented notations will represent the overall design and the architecture of directory servers. Also any association, data flow and other user interfaces if present, has to be defined at this stage. From the environment architecture the modeling embedded system and software design modeling represents this. Any system needs to be modeled before it is built and also these models should be converted to a notational diagram. Thus these directory server, conference managers and other components and connectors of this system are first modeled and represented using notations.

Using the basic notations we represent these directory servers as follows and if these directory servers contain one or more conference managers then it will be referred to as a Module Directory server.

These components establish, maintain and remove connections between conference participants. They are also responsible for sending multicast messages to the conference agents that will be defined in the next section. These conference managers participate in the conference with other conference objects. Conference managers are also represented using component objects.

These conference agents handle commands entered from the participant’s workstation. They are responsible for formatting and transmitting messages to their corresponding conference managers. These conference agents communicate with one another indirectly through their corresponding conference managers.

6.3. Component interaction

Once the objects of the system are defined, identifying the active and passive objects are important. With the method provided we identify the passive and active objects. These are represented in the notational diagram. Once all the objects and their components are defined, we need to identify or present the interaction between these components in a clear way.

The component interaction gives overall understanding of the system. These are usually represented using connectors and we provide many different types of connectors based on the types of data flow.

Interaction of conference agent, conference managers and directory servers varies depending on the state of a conference. For example, when scheduling a conference only directory severs and conference managers are involved. When a conference manager receives a request to
schedule a conference, it interacts with its local directory servers to obtain location of the invited participants.

![Software Architecture for Multimedia system](image)

Figure 6. Software Architecture for Multimedia system

7. Conclusion

Real time systems are becoming a pervasive element of society. Many methodologies for these systems will be made a significant progress in near future. This paper has contributed by presenting a new guidelines and general notations for building real time systems. This paper used object-oriented approach to both identify and define different types of objects and their interfaces. We exploited the object-oriented concepts to build a new set of notations which is then used to design an architectural structure.

The guidelines and procedures for building architectural design in form of a flow chart format have been used to show that a new architecture can be designed. With the presence of the case study – Multimedia real time conferencing system, the paper is complete as it proves the validity of the software architecture.

We are going to work on the answers to some of the most ambiguous questions that may arise in the near future and to implement an environment where we can use procedures and notations to formulate architecture for specific real time systems.

References

[9] Chung, Jen-Yao An industry view on services-oriented architecture and web services, IEEE International Workshop on Services-Oriented System Engineering, 2005, pp. 59

Authors

Dr. Rhan Jung is currently a professor in the Department of Computer Engineering, Kangwon National University in Korea. She received her Ph.D degree in Computer Science from Catholic University of Taegu in 1998 and MS degree in Computer Engineering from Chung-Ang University 1983. She had been a visitor research staff in Computer Science Department of UMASS from December 1999 to June 2001. She is a member of KISS and KIPS. Her research interests are Component Based Development, Software Architecture, and Software Process.

Dr. Soung Won Kim is currently a professor in the Department of Electrical & Electronics Engineering, Anyang University in Korea. He received Ph.D degree in Computer Science from Catholic University of Taegu in 2000 and MS degree in Electrical Engineering from Dan-Kook University 1982. He is a member of KISS and KIPS. He research interests are Component Based Reuse, Software Engineering, and Computer Graphics.