

10-12-1999

Domain specific architecture development for enterprise systems based on common object request broker architecture (CORBA)

Vidya G. Bhat

Florida International University

Follow this and additional works at: <http://digitalcommons.fiu.edu/etd>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Bhat, Vidya G., "Domain specific architecture development for enterprise systems based on common object request broker architecture (CORBA)" (1999). *FIU Electronic Theses and Dissertations*. Paper 1673.
<http://digitalcommons.fiu.edu/etd/1673>

This work is brought to you for free and open access by the University Graduate School at FIU Digital Commons. It has been accepted for inclusion in FIU Electronic Theses and Dissertations by an authorized administrator of FIU Digital Commons. For more information, please contact dcc@fiu.edu.

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

DOMAIN SPECIFIC ARCHITECTURE DEVELOPMENT FOR ENTERPRISE
SYSTEMS BASED ON COMMON OBJECT REQUEST BROKER ARCHITECTURE
(CORBA)

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Vidya G. Bhat

1999

To: Dean Arthur W. Heriott
College of Arts and Sciences

This thesis, written by Vidya G. Bhat, and entitled Domain Specific Architecture Development for Enterprise Systems based on Common Object Request Broker Architecture (CORBA), having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved.

Paul Attie

Raimund Ege

Yi Deng, Major Professor

Date of Defense: October 12, 1999

The thesis of Vidya G. Bhat is approved.

Dean Arthur W. Heriott
College of Arts and Sciences

Dean Richard L. Campbell
Division of Graduate Studies

Florida International University, 1999

ACKNOWLEDGMENTS

I would like to thank Dr. Yi Deng, my advisor, for having the faith and confidence in me to do this project and for his constant guidance and support throughout the thesis. I would also like to express my gratitude and appreciation to the following people:

To my thesis committee members, Dr. Ege and Dr. Attie. To Kent Wreder, Eric Novarro, Eric Butler and everyone else at Baptist Health Systems for all their answers to my unending questions, their time and patience. To my colleagues and friends at CADSE George McGivan, Luis Espinal, James Goolsby, Nikhil Iyer and Jinny Uppal for their constant support, motivation, encouragement and for reading through my thesis for all those corrections. To all my friends here in Miami for being my personal cheerleading squad. To my parents and brother for their constant support and their trust in me. To Karthik Madhyanapu, for always being there for me.

ABSTRACT OF THE THESIS
DOMAIN SPECIFIC ARCHITECTURE DEVELOPMENT FOR ENTERPRISE
SYSTEMS BASED ON COMMON OBJECT REQUEST BROKER
ARCHITECTURE (CORBA)

by

Vidya G. Bhat

Florida International University, 1999

Miami, Florida

Professor Yi Deng, Major Professor

Large business organizations with enterprise wide systems have followed an ad hoc incremental growth pattern. They are either monolithic, that are difficult to replace and maintain, or are components with little or no interoperability between them. Such systems suffer from lack of uniformity and definition in their information technology infrastructure. To migrate from this state, to systems that are extensible, interoperable and non-redundant in functionality it is very important to focus on the architecture. We use the healthcare enterprise system as a case study for the purpose of this thesis. It is indeed difficult, if not impossible to construct the overall architecture of the system without identifying the individual components of the system. Hence we follow an incremental methodology in identifying and developing each component. One such component is Order management, which is an essential component of a healthcare information system that offers enterprise wide functionality.

TABLE OF CONTENTS

CHAPTER	PAGE
I. INTRODUCTION.....	1
1.1 PROBLEM STATEMENT	1
1.2 THE SOFTWARE DEVELOPMENT PATH	3
1.3 CADSE - BHS PROJECT	4
1.4 CAUSES OF THESE PROBLEMS	5
1.5 CONTRIBUTION OF THE THESIS	6
II. BACKGROUND.....	7
2.1 SOFTWARE ARCHITECTURE.....	7
2.2 WHY IS SOFTWARE ARCHITECTURE IMPORTANT?.....	11
2.3 COMPONENT-BASED SYSTEM DEVELOPMENT.....	13
2.4 LARGE-SCALE ENTERPRISE-WIDE SYSTEMS	14
2.5 DOMAIN SPECIFIC ARCHITECTURE.....	16
III. STATE AND ISSUES OF HEALTHCARE INFORMATION SYSTEMS	18
3.1 DOMAIN ANALYSIS	19
3.2 HEALTHCARE ENTERPRISE SYSTEMS.....	20
3.3 CURRENT INFRASTRUCTURE IN HEALTHCARE.....	22
3.4 THE CORBAMED EFFORT	28
IV. ORDER MANAGEMENT	31
4.1 SIGNIFICANCE OF OM	33
V. PROBLEMS AND THEIR CAUSES	35
5.1 PROBLEMS IN THE ENTERPRISE	35
5.2 CAUSES OF THESE PROBLEMS.....	37
5.3 OUR APPROACH	40
VI. PROPOSED SOLUTION	44
6.1 THE ORDER MANAGEMENT BUSINESS WORKFLOW	44
6.2 USE CASE FOR ORDER MANAGEMENT	45
6.3 THE INFORMATION VIEWPOINT OF ORDER MANAGEMENT	58
VII. AN ENTERPRISE VIEW	73
7.1 SOLUTION: A 'GOOD' ARCHITECTURE	73
7.2 ORDER SERVICE.....	74
7.3 ARCHITECTURE FOR AN ORDER SERVICE.....	81
7.4 SUMMARY	82
VIII. CONCLUSION	86
FUTURE ISSUES/ POTENTIAL.....	87
LIST OF REFERENCES	89
APPENDICES.....	94

LIST OF FIGURES

FIGURE	PAGE
FIGURE 1 THE PROBLEM OF INTEGRATING COMPLEX SOFTWARE.....	15
FIGURE 2 DEVELOPMENT PROCESS	19
FIGURE 3 INTERFACE ENGINE ARCHITECTURE FOR SYSTEM INTEROPERABILITY.....	24
FIGURE 4 THE CHANGING PARADIGM OF HEALTH INFORMATICS	29
FIGURE 5 BUSINESS PROCESSES RELATED TO ORDER MANAGEMENT.....	33
FIGURE 6 MOVING TOWARD STANDARD-BASED ARCHITECTURE.....	42
FIGURE 7 USE CASE FOR ORDER MANAGEMENT.....	49
FIGURE 8 SUB-USE CASE FOR PLACE ORDER	54
FIGURE 9 INITIAL DATA MODEL FOR ORDER MANAGEMENT.....	59
FIGURE 10 CONCEPTUAL VIEW OF ORDER MANAGEMENT	61
FIGURE 12 DEPENDENCY DIAGRAM FOR ORDER SERVICE.....	75
FIGURE 13 SEQUENCE DIAGRAM FOR PROCESSING A RADIOLOGY ORDER	78
FIGURE 14 CREATE AN ORDER	79
FIGURE 15 GENERATING CHARGES AFTER AN ORDER IS PLACED	80
FIGURE 16 NOTIFICATION AFTER PATIENT DISCHARGE	80

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

Many of today's corporate systems were developed and/or acquired in a piecemeal fashion, incorporating a component on a per need basis. A large-scale organization often possesses multiple fragmented applications. Each software system performs some limited range of useful functions, but the systems do not interoperate effectively, and very often duplicate functionality. Integration between the components is based mostly on proprietary solutions. Such ad hoc integration usually produces undocumented brittle systems that are expensive to maintain and costly to upgrade. Such stovepipe or monolithic systems also resist adaptation to changes in user requirements, business processes and commercial technology. Poor integration leads to substantial organizational inefficiencies such as redundant data entry, unnecessary and ad hoc data conversion and file transfers, and inability to effectively integrate and utilize enterprise-wide data and knowledge. The result is redundancy in information and poor functionality among the systems.

Systems in large business organizations have followed an incremental growth pattern and are either monolithic (difficult to replace and maintain) or consist of components with little or no interoperability between them. Integration of these components to achieve business objectives has often been inefficient. These large-scale systems suffer from lack

of uniformity and definition in their information technology infrastructure. Changing requirements (user and business processes) are the major cost drivers in software development and maintenance [Hor93]. Due to a lack of overall software *architecture* and a well-defined business model, the infrastructure inhibits future evolution and adaptability to change. As a result, maintainability becomes a major issue in these systems.

To migrate from this state, to systems that are extensible, interoperable, maintainable and non-redundant in functionality it is very important to focus on software architecture. Systems with "good" architecture [4] (i.e. having well defined modules, well-defined interfaces and uniform interaction patterns) can alleviate and/or avoid the drawbacks stated above. To achieve enterprise-wide systems that are open, interoperable and integrable, system development must follow a uniform architectural model for composing and integrating different systems, and is based on established and accepted technology to ensure long term stability and extensibility. OMG's (Object Management Group) Object Management Architecture (OMA) and Common Object Request Broker Architecture (CORBA) provide a common architectural solution to support component-based integration and evolution under the notion of distributed objects. But this architecture is not sufficient to support domain-specific functionality. Domains usually have various specific functionality and requirements. Applying architectural techniques at a domain level expands the reuse potential of architectural components to multiple, related systems. Architectures play an important role in facilitating domain-specific reuse by defining implementation frameworks for constructing systems within a domain [x].

We contend that an overall infrastructure (as it relates to basic services and interfaces between software components on which applications can be developed and integrated) for any domain based on OMA/CORBA (i.e. healthcare) will solve these problems and issues. The scope of this thesis is to identify and model a component within healthcare that contributes to the overall healthcare architecture framework. It proposes a service for this component and illustrates that several ills faced by the current systems can be alleviated.

1.2 The Software Development Path

Because of the success of network computing concepts and web-based Intranets, legacy software components are often required to be integrated into new and novel configurations. This has facilitated enterprises to inter-connect previously isolated business domains and use tools to streamline business processes, reduce administration efforts and decrease operating costs. This has in turn resulted in maximizing productivity. But very often different business domain systems have been inter-connected through customized proprietary solutions whenever it has been required. This has resulted in stovepipe systems that are monolithic, lack discernable software architecture, lack provision for reuse and extension and are expensive to maintain. Enterprises that have built generations of loosely (or uncoupled) information systems are now scrambling to integrate them into an organic whole.

This involves the integration of components drawn from disparate systems. This introduces problems that are referred to as ‘architectural mismatch’ [GAO95]. This refers to situations in which software components are resistant to integration because they exhibit one or more forms of incompatibility, or “mismatch”.

1.3 CADSE - BHS Project

The CADSE (Center for Advanced Distributed Software Engineering) – BHS (Baptist Health Systems) project is an effort to address the problem of integrating disparate systems together such that migration of existing systems to systems that are open and more interoperable is simplified. The purpose is to collaboratively develop an architecture-centered methodology and domain specific system architecture for the development of the next generation distributed enterprise systems by combining latest software engineering research and industry-wide standard OMA/CORBA. The goal of the research is not only to develop a uniform architectural model for enterprise system development and integration, but also to advance the state of knowledge and research on complex system development. For the purpose of this research we consider the healthcare domain and the large complex systems within this domain.

BHS (Baptist Health Systems of South Florida) is a large health care organization comprised of five major hospitals, multiple outpatient clinics and multiple physician offices throughout South Florida. A large healthcare organization is composed of multiple smaller distinct business units and each may require a system provided by one or more different vendors. Each individual system provides functionality e.g. access to

patient demographic and admission information, create, manipulate and access patient medical record, and some form of workflow engine that support the business flow within the department. The objectives are to integrate data and functionality from multiple clinical and business systems such that the enterprise business process is optimized as a whole. Such an enterprise healthcare information system is commonly referred to as a Computerized Patient Record (CPR).

In the past, healthcare systems were developed or acquired in a piecemeal fashion based on individual business needs and integrated [often] using proprietary solutions. Although each individual system maybe put in place based on a well defined and planned process based on departmental business goals, collectively from enterprise (or CPR) point of view, the systems are acquired in a disparate fashion and tightly coupled together through ad hoc means. The problem is further intensified by the merge of hospitals and healthcare organizations, each of which comes with its own set of systems that are normally not interoperable. These together have resulted in stovepipe systems that have many duplicated functions and are monolithic, non-extensible and not interoperable.

1.4 Causes of these problems

The thesis examines in detail, different causes of these problems. There is no underlying infrastructure in building an enterprise-wide information system in large-scale industries like healthcare. Lack of an explicit data model for the components within the system makes integration of these components a difficult issue. There is no predictable interaction pattern between the components as also there is a lack of definition of their

behavior. There is also no concept of a global architecture for the enterprise, especially in the healthcare domain. These causes are explained in detail in chapter 5.

1.5 Contribution of the thesis

The present work represents a step on a road towards a comprehensive architecture for open, modular, patient-based health information and communication systems. There is a need to record the analysis and design information for further use and research since the application of software architecture principles to the healthcare domain is very new. This thesis is a record of such information required to further a service-based architecture in a particular domain i.e. Healthcare. Moreover, the concept of software architecture has been previously applied to single systems but hardly ever to large-scale enterprise systems [GAO94]. Some work done in large-scale environment is strictly confined to domains based on sound engineering principles and where application of information technology is advanced [ABH93] [Cle95] [Cza97].

CHAPTER 2

BACKGROUND

2.1 Software Architecture

The field of software architecture has only recently emerged as an explicit focus for research and development, and there is not yet a [standard] universally accepted definition for the term. In spite of that, the importance of software architecture in government, industry and academia is widely acknowledged. This emergence of its importance is evidenced by a large body of recent work in areas such as module interface languages, domain specific architectures, architectural description languages, formal underpinnings for architectural design, and architectural design environments [CN96] [GP94]. In addition, an implicit body of work exists in the form of descriptive terms used informally to describe systems. But there is no well-defined terminology or notation to characterize architectural structures. Good software engineers commonly use architectural principles that have emerged as rules of thumb or patterns informally over a period of time. Some other principles are carefully documented as industry or scientific standards.

The term architecture is commonly used in many different ways; some of them are [GP95]

- (a) the architecture of a particular system, (e.g. “the architecture of this system consists of the following components”),
- (b) an architectural style (e.g. “this system adopts a client-server architecture”),
- (c) the general study of architecture (e.g. “the papers in this journal are about architecture”).

Within software engineering, most uses of the term “software architecture” focus on the first of these interpretations. The recent emergence of interest in software architecture has been prompted by two distinct trends. The first is the recognition that over the years designers have begun to develop a shared repertoire of methods, techniques, patterns and idioms for structuring complex software systems. The structure of software has long been recognized as an important issue of concern. For example, the box and line diagrams and explanatory prose that typically accompany a high-level system description often refer to such organizations as a 'pipeline', a 'client-server system' etc. These terms permit complex systems to be described using abstractions such that it can be interpreted in a intelligent fashion. Moreover, they provide significant semantic content that informs others about the kinds of properties that the system will have, the expected paths of evolution, its overall computational paradigm, and its relationship to similar systems.

The second trend is the concern with exploiting specific domains to provide reusable frameworks for product families. Such exploitation is based on the idea that common aspects of a collection of related systems can be extracted so that each new system can be built at relatively low cost by instantiating the shared design. Familiar examples include

the standard decomposition of a compiler, standardized communication protocols, fourth generation languages etc.

We claim that most of the problems faced by enterprise wide systems is due to lack of architecture and architectural vision during development of the subsystems and overall systems. A critical aspect of any complex, large-scale enterprise-wide system is its architecture [GAO94]. At an architectural level of design a system is typically described as a composition of high-level, interacting components.

Software Architecture is a field that has generated widespread interest and has grown in importance not only in academia but also in the industry. As the size and complexity of software systems increases, the design problem goes beyond the algorithms and data structures of the computation: designing and specifying the overall system emerges as a new kind of problem. Some of the issues involved in the structure of systems include gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; composition of design elements; scaling and performance; and selection among design alternatives [GS94]. This is considered the software architecture level of design.

What exactly, then is software architecture? There is no standard universally accepted definition of the term, for software architecture is a field in its infancy, although its roots run deep in software engineering. Though there is no standard definition, there is also no shortage of different ideas and concepts for the term. According to Bass et al, in

‘Software Architecture in Practice’ [BCK97], the software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them. “Externally visible” properties refer to the assumptions other components can make of a component, such as its provided services, performance characteristics, fault handling, shared resource usage etc.

Architecture defines components. The architecture embodies information about how the components interact with each other. Architecture specifically omits content information about components that does not pertain to their interaction i.e. the components are expected to encapsulate all information except that which is required to interact with other components. Systems can comprise more than one structure and the architectural component can be anything from an object, a process, a library to a commercial product; all these and more. This also means that every software system has an architecture, because every system can be shown to be composed of components and relations among them. The behavior of each component is part of the architecture, as long as that behavior can be observed or discerned from the point of view of another component. This behavior is what allows components to interact with each other, which is clearly part of the architecture. More details of this concept and its applicability to the present work is dealt with in detail in the next chapter.

2.2 Why is software architecture important?

Architectural design of large systems has always played a significant role in determining the success of a system. Choosing an inappropriate architecture can have a disastrous effect on the final system. A principled use of software architecture can have a positive impact on the following aspects of software development. [GP95]

- **Understanding:** Software architecture simplifies our ability to comprehend large systems by presenting them at a level of abstraction at which a system's high-level design can be understood. A good description of the overall system architecture exposes the high-level constraints on system design and helps provide the rationale to make specific architectural choices.
- **Reuse:** Architectural descriptions support reuse at multiple levels such as in the form of component libraries, reuse of large components and frameworks into which components can be integrated.
- **Evolution:** Software architecture can expose the dimensions along which a system is expected to evolve. Architectural descriptions can also separate the functionality of a component from the way it interacts with other components. This facilitates changes to the interaction mechanism whenever there are evolving concerns about performance, interoperability, prototyping and reuse.
- **Analysis:** Architectural descriptions provide new opportunities for analysis, including high-level forms of system consistency checking, conformance to an architectural style, conformance to quality attributes, and domain-specific analyses for architectures that conform to specific styles [GP94].

- Management: During the development of industrial software, achieving a viable software architecture is a key milestone and this involves specifying a software's initial requirements, its anticipated growth dimensions etc. And if these conditions are not satisfied then there is a significant risk that the system will be either inadequate or unable to accommodate change.

Software architecture has a broad impact on market drivers that are important for software-intensive businesses. Software has become an integral part of a wide variety of products and many of these products have been out in the market for some time. Hence, there is a broad base of existing and well-tested software. These products involve significant investment in terms of time and money and the reuse of these software products in different environments would be of fiscal importance to software producers. Software architecture, to some extent focuses on domain-specific abstractions and use/reuse of various existing software elements at varying degree of granularity and in doing so, impacts one of the market drivers.

A skeletal framework is needed to achieve conceptual integrity. Conceptual integrity in terms of software architecture means unity in design and uniformity in terms of the underlying concepts and paradigms [1]. This infrastructure design will be used to identify the decomposition of the overall functionality into various subsystems, act as a guideline for developing the systems and used for integrating various elements of the system. This infrastructure can also be used to unite and leverage various existing applications and systems while providing flexibility to adopt best-of-breed solutions. Software architecture can effectively establish a common framework across domain-related products and thus

aid in achieving interoperability between components across product lines. This is another market driver that adds to the successful exploitation of a company's software products. Software development is moving significantly in the direction of component-based development, which means that significant portions of software systems are procured instead of being produced. Large systems can be rapidly built and deployed by buying/importing externally developed components.

2.3 Component-Based System Development

A component is an executable unit of code that provides physical black box encapsulation of related services. Its services can only be accessed through a consistent, published interface that includes an interaction standard. A component must be capable of being connected to other components, through a communications interface, to form a larger group that serves a larger set of functionality. Components provide an effective granularity of reuse through consistent published interfaces that encapsulate the implementation.

Components can be defined and understood at different levels of granularity. At the lowest level, components can be program (object) modules where as at the very highest level they can be complete systems that provide certain functionality. Here we consider components as being various subsystems within an enterprise. For example, a healthcare organization consists of several independent subsystems serving specific purposes. From the enterprise point of view these are components. Development in the field of software architecture is shifting the focus to assembling components that were built independently

from each other. And this focus is justified as far as enterprise systems are concerned. Enterprise systems consist of large-scale systems consisting of numerous independently developed systems especially for different purposes. From an end-user's perspective, the focus is on 'putting' these systems together. This is the view that we take during the course of this work.

2.4 Large-scale enterprise-wide systems

Within a typical enterprise, there are many applications that have been developed without considering how the applications will interoperate. This applies to both custom and commercial applications. For example, some commercial off-the-shelf (COTS) applications provide an all-encompassing set of functionality to the users while they are working within the application. The application source code is seldom available from the vendor, and even if it is available, it is undesirable to modify it due to support costs. In most applications, data are stored in a unique, proprietary format with limited conversion capabilities. Even the more advanced applications within an organization, huge custom applications incorporating hundreds of classes and developed using state-of-the-art GUI builders and automatic code generators, only provide interoperability between applications using the same tools and framework classes. The goal of interoperability is further compromised by the existence of long-lived legacy applications that provide some degree of custom capability within an organization. Often, these legacy applications have no application program interface (API) for use by other applications. If required to communicate they do so using low-level communication mechanisms, such as, files, Dynamic Data Exchange (DDE) or an operating system specific protocol. Within a

distributed computing environment, this problem is compounded by the interconnectivity of the environment where the number of applications, data formats, and communication mechanisms is so numerous that no single application can be prepared to handle all of the various mechanisms interacting within the environment.

A typical situation in many large-scale organizations is depicted in Figure 1.

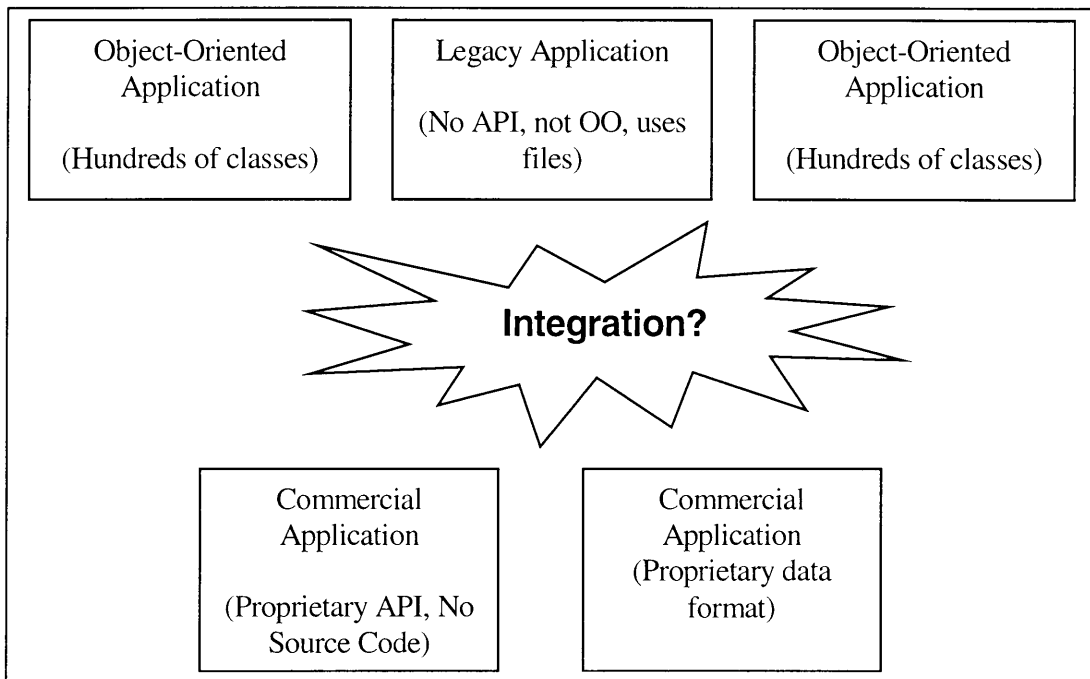


Figure 1 The problem of integrating complex software

There are several groups of applications developed with different technology and if they interoperate at all it is through mechanisms only available in small subset of applications within an organization. Some of these systems are legacy systems, which have been a part of the corporate infrastructure since some time and they perform their tasks well. There is little or no incentive to re-engineer and/or redevelop them. There are other

systems based on component technology (COTS packages) and contain proprietary API's with no access to source code. Often, they use RPC's, or whatever was the popular communication technique at the time of their development. Object-oriented applications are developed using different tools, class libraries, and frameworks, and provide limited, if any, means of communicating with applications not sharing identical set of components. A general-purpose method of integrating software systems seemed elusive to the industry, and most of the developers/ end-users have settled for point-to-point integration devoid of an overall architecture to guide system evolution.

A standard middleware like CORBA provides a better solution to the recurring problem of software integration and interoperability. The OMA/CORBA architecture can be used as an underlying infrastructure to develop as well as integrate distributed software and applications.

2.5 Domain Specific Architecture

There is research being done in the field of domain-specific architecture [Cza97]. In this case, the research on software architecture is specific to a particular domain or a group of systems that have something in common. Having a sound architecture for a large-scale system makes for better development process and results in a better product. If this architecture is enhanced by domain specific requirements, it creates a better model for reuse and product development. Time to market for new services is crucial for survival and prospering in a highly competitive market. This situation is observed in domains like Telecommunication, healthcare and several similar domains. In order to cut development

time and cost, the vision of component-based development has to be pursued. A domain specific architecture provides a common framework for component interoperability within a domain.

This thesis contributes to a domain specific architecture based on a standard technology, CORBA. Having such a domain-specific architecture for a particular domain enhances the advantages the CORBA provides for system interoperability and integration.

CHAPTER 3

STATE AND ISSUES OF HEALTHCARE INFORMATION SYSTEMS

Our example of a large-scale enterprise system is the Healthcare information system. Enterprises in Healthcare domain suffer from similar problems as any large-scale complex enterprise. Our analysis of the domain revealed several problems, which is discussed below. Our analysis is based on extensive review and research conducted with the help of BHS.

Baptist Health Systems of South Florida (BHS) is comprised of four acute care hospitals as well as various other units. Each of these hospitals is huge and serves more than a hundred thousand patients every year. More than one hundred and fifty information systems serve the entire enterprise and the annual budget on information systems is approximately fifteen million. All these information systems process more than ten terabytes of data per year with the amount of data increasing every year. This puts the enterprise wide information systems in BHS under the category of large-scale information systems. To study the systems in detail and understand them, the first step in the thesis was domain analysis.

3.1 Domain Analysis

Domain analysis (first introduced in the 1980s) is the process by which information used in developing systems in a domain is identified, captured, and organized with the purpose of making it reusable when creating new systems [Pri90]. Analysis is essential to understand the domain as well as the information relevant to the domain. This was the first step in understanding the requirements and specifics of the healthcare domain. Analysis was carried out by a series of interviews and meetings with the experts in this domain from BHS. They provided us with the basic business workflow of a typical Healthcare Facility (HCF). These results are discussed in detail in the following chapters. We followed the development process showed in Figure 2.

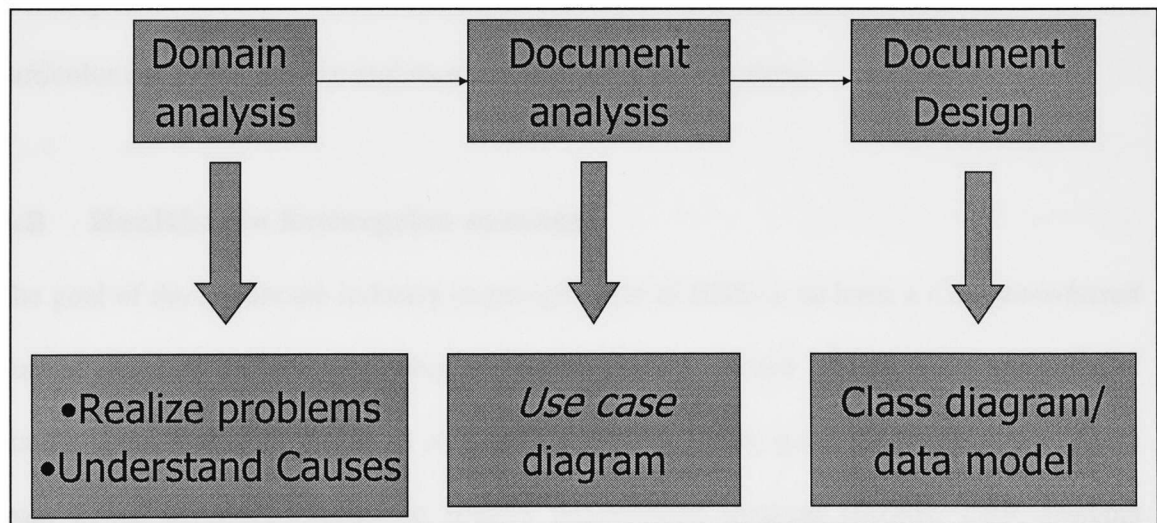


Figure 2 Development Process

The initial part as described in this section includes domain analysis and understanding the problems and their causes. The result of this domain analysis is documenting these

problems and their causes. This is captured in Chapter 5: Problems and their causes. The other two steps of the process are covered in Chapter 6: Proposed Solution.

Along with the interviews and meetings, we studied the current systems deployed at BHS. In particular, I studied the Order Management System, a system that manages orders within a HCF. This system is described in detail in Chapter 4. Lack of any detailed document for the design of this system hindered the understanding. This is one of the common problems encountered during system integration and is responsible for a number of mismatches [GAO95]. My study of the system not only helped me understand how the current systems are deployed and how they exist but it also revealed several problems that they face. We discuss the healthcare industry in general and then shift our focus to a particular enterprise system and its problems in the next section.

3.2 Healthcare Enterprise systems

The goal of the healthcare industry (especially that of BHS) is to have a *Computer-based Patient Record* (CPR). A computer-based patient record (CPR) is electronically maintained information about an individual's lifetime health status and health care. Apart from being automated forms of today's paper-based medical records, CPR systems facilitate the capture, storage, processing, communication, security, and presentation of non-redundant health information [i].

Our analysis studied the current practice, which we discuss in detail below moving from the enterprise level to specific system level. Healthcare is a complex domain with varied

and vastly differing yet inter-related business processes. Hence, the information and computer systems used in this domain are also complex and very often play critical roles in the business processes. They link together geographically distributed hospitals, clinics, physician offices and other business units with distinct business functions. And as in any other domain with a number of varied business processes using information systems, this domain too suffers from all the problems and more: the reason being that information technology in the healthcare domain is still in its infancy and very immature unlike other domains such as Telecommunications.

A large healthcare organization is composed of multiple smaller distinct business units like Radiology, Laboratory, Pharmacy etc. They interact with either one or a small select handful of systems to perform their job functions. A multitude of systems exists in the marketplace to select from in an attempt to automate departmental areas. Each has varying cost and effectiveness and most of them are mature and perform well within their own distinct business unit. A single vendor solution for all these varied needs is often impractical and inefficient. The best choice is to opt for best of breed systems based on a vendor's ability to meet user needs. System acquisition in BHS has been ad hoc, done on a per need basis. This has been the choice of most large healthcare organizations but has unfortunately resulted in monolithic systems that are neither flexible nor extensible and has resulted in either vendor lock-in or expensive custom programming required in getting these disparate systems to work together. Systems are built in an incompatible fashion by vendors without concern to preserve or even allow an overall well-defined architecture for the organization [BNW98].

Another issue is the merger of hospitals. BHS is made up of several hospitals acquired one after another. Each hospital brings with it a complete set of information systems that perform functions identical to the already existing systems introducing loss of integrity. Systems being integrated across hospitals or healthcare facilities are a far off issue; systems within a single hospital are not integrated efficiently. Several standards exist within healthcare to ease interoperability between systems and hence aid in system integration (DICOM, CCOW, HL7) [iii] [iv] [v]. However, most of these standards are message-based, since they were meant for systems that communicated via explicit messages.

The approach based on these message-based standards (HL7) to achieve interoperability is explained in detail in the next section where some of the problems in the current practice are also discussed.

3.3 Current Infrastructure in Healthcare

The current practice in Healthcare systems is defined with reference to the issues and problems summarized at the end of this chapter. Systems need to be interoperable to exchange information. Integration of systems within the healthcare facility to achieve interoperability, in particular in BHS, is achieved through several different means.

A point-to-point connection achieves a level of interoperability enough to aid basic exchange of information but creates a number of problems. This is basic communication

level of interoperability. The number of point-to-point connections could blow up exponentially as the number of systems increase. Designing and developing each point-to-point interface requires vendors, sometimes competing vendors, to work closely with each other. The interfaces are very expensive to develop and they do not scale in terms of functionality. If a systems requirements change, or a system is replaced, new interfaces must be developed. Taking an example from Baptist Hospital, there is a point-to-point interface between CareFlowNet (transcription system) and ClinStar Patient Care System for Orders/results where communication takes place through HL7 messages.

Another solution uses a central server to do all the interfacing. This central server, also called *interface engine* is responsible for data translation and routing between a source and one/more destination(s).

The interface engine consists of two main components: the server, which receives messages, performs needed translations, and routes messages to their destinations; and communication clients, which carry messages to and from the server.

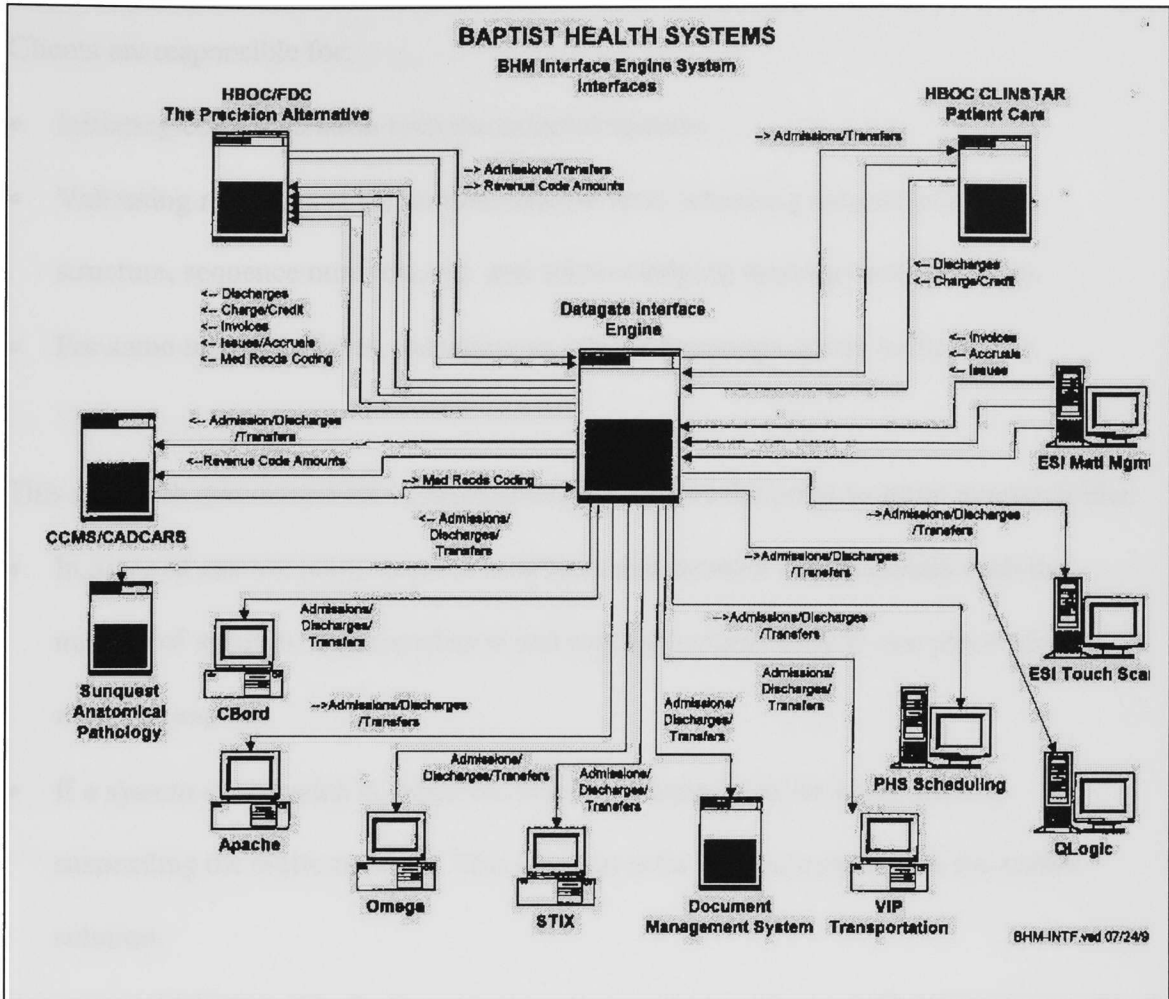


Figure 3 Interface Engine Architecture for system interoperability

The server's responsibilities are:

- Receiving and validating messages from the communication clients
- Translating and routing messages
- Managing message queues to outbound communication clients

The server is connected to the communication clients (basically, a process) which in turn connects to the external systems. These systems are the sources and destinations of all

messages¹. Clients connect to the external system through a network or serial port.

Clients are responsible for:

- Initiating communication with the external systems.
- Validating messages at the communication level (checking data stream block structure, sequence numbers, etc. and acknowledging message receipt status).
- For some inbound clients, managing an inbound message queue to the server.

This approach alleviates a lot of disadvantages faced by the point-to-point approach like:

- In place of custom point-to-point interfaces that increase exponentially with the number of systems, data translation and routing happens only in one place: the interface engine.
- If a system is upgraded or replaced, interface changes can be made without suspending the entire network. This solution is far less expensive than the earlier solution.

In spite of several advantages that the interface engine approach has over the traditional point-to-point approach, many systems in the healthcare environment are currently integrated using the point-to-point approach. The biggest disadvantage with this approach is system evolution and flexibility. Though the approach of using interface engine seems to solve several problems, it has drawbacks of its own, such as:

¹ Message here means either a stream of data, a batch file or an HL7 message i.e. a message in HL7 standard format.

- The Interface Engine (IE) just routes messages from source to destination, with some translation, if required. In essence, it does not give query-retrieve capability to any system using it. It is either configured to send information from one source to destination or it is not.
- It does not allow for systems to share functionality, though it allows sharing of data. This creates redundancy in functionality across systems.
- There is also redundancy of information or replication of data.
- There is no way to know if the destination system has processed the data passed to it. E.g. suppose system 'A' sends a message to system 'B'. Though system B receives data, it is not able to process it due to various reasons. For example, this happens when system 'B' does not understand the data that it received and it simply discards the data. There is no way to indicate to 'A' that the information has not been used and that the data may need to be resent. This may result in inconsistencies in data across systems.

Many of the systems used in the healthcare environment are adopting the HL7 messaging standard. HL7 was founded in 1987 to develop standards for the electronic interchange of clinical, financial and administrative information among independent health care oriented computer systems; e.g., hospital information systems, clinical laboratory systems, enterprise systems and pharmacy systems [iii]. Systems using this standard use HL7 standard messages to communicate with each other. HL7 being a messaging standard is very good for data transfer but it does not capture the functionality of the system. It also defines extra fields at the end of each message template, which can be used as desired by

the implementing vendor. These extra fields were placed to give flexibility but it also introduces the problem of inconsistencies in the message format as implemented by two different systems and incompatibility in data models when systems are integrated.

Either means (using IE or point-to-point) to achieve interoperability between these systems does not reflect business logic. Each of these systems has its own proprietary data model. The resulting environment has multiple incompatible data models. Each system then makes assumptions about the other systems data model. This creates data and functional redundancy.

Systems are proprietary; hence they are not 'open'. Due to vendor lock-in, systems are not easily adaptable. Since the system is not open, a user organization (like BHS) has to depend on the vendor to make changes to the system, if desired or if the user requirements change. This could be expensive in terms of cost and time.

There is a lack of uniformity in design and overall infrastructure, which is other words, is a lack of conceptual integrity. This makes system integration difficult and inefficient. Rapid deployment is not possible due to lack of uniform underlying infrastructure. Hence, there is a need to have a basic infrastructure, an architecture that is domain specific to healthcare. Our approach to this is based on a standard architecture framework, CORBA. As mentioned earlier, CORBA solves many of the problems faced with system integration. Our efforts are aligned with those of the CORBAmed group.

3.4 The CORBAmed effort

CORBAmed is the Healthcare Domain Task Force of the OMG, the Object Management Group, a non-profit international organization indented to the promotion of Object Oriented methodologies. Task Forces deal with the specific aspect of various application domains, which have common interest in the same interface technologies. Task Forces are specialized in various domains as Electronic Commerce, Manufacturing, etc. and CORBAmed in health care applications.

CORBAmed defines standardized interfaces to many healthcare "Object Oriented Services", across most usual platforms, and available in the public domain. One of the goals of CORBAmed is to enable immediate significant achievements to be achieved by clearly defining the scope and boundaries of, and the relationships between the components in, one or more sub-sections of the vast domain of healthcare.

The business case for the healthcare industry is that participating institutions need to interoperate by sharing their information; but as individual business entities, each institution in an Integrated Delivery System (IDS) or hospital must maintain ownership of their important patient-centered records. Neither centralized systems (like mainframe systems) nor client-server systems can meet these needs. However, *distributed object technology* seems ideal for this purpose. The object oriented (OO) principle of encapsulation is ideal for the protection of data ownership while allowing controlled access to the information by external clients. Distributed object technology (such as CORBA) allows healthcare related objects to communicate over a network; in particular

across physical computer boundaries. CORBA, specifically, as a platform and language independent standard for distributed object technology, seems to offer the best migration path from the current systems to interoperable IDS's [Roadmap].

This paradigm shift in healthcare is seen to be toward "horizontally oriented" (across the enterprise) business objects rather than the current "vertical oriented" departmental systems. The figure below gives an idea of this trend.

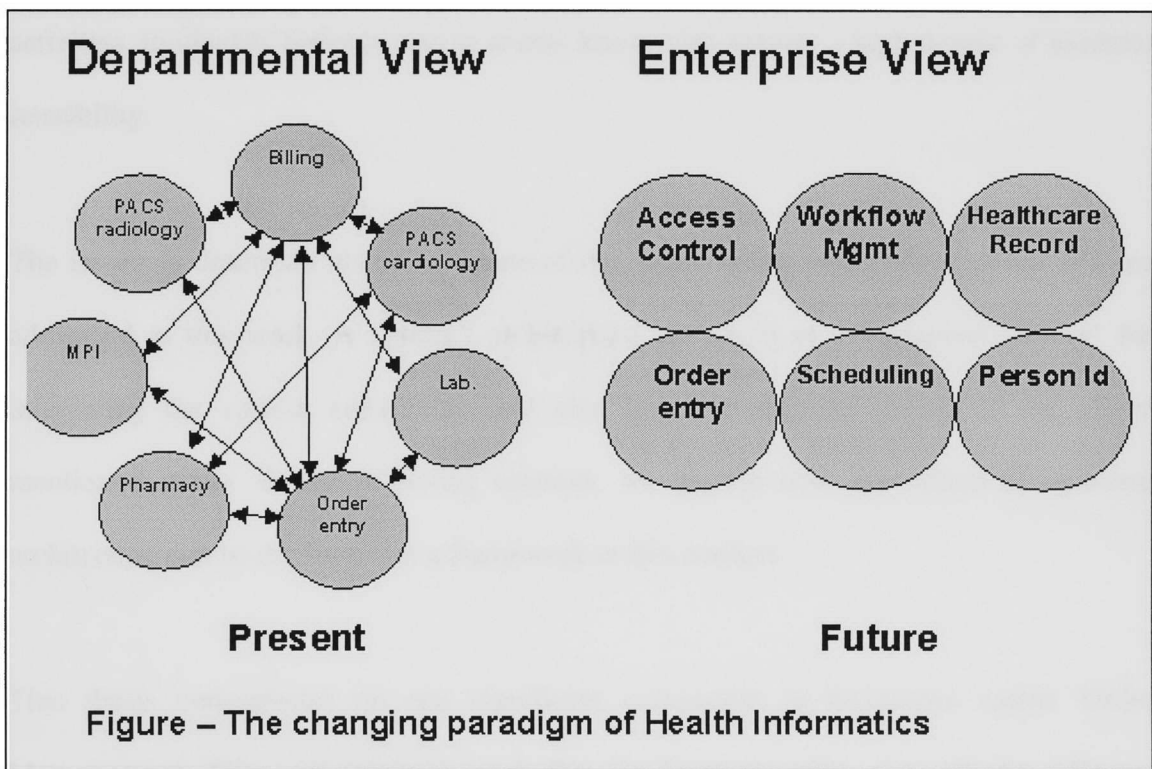


Figure 4 The changing paradigm of Health Informatics

Instead of viewing the IDS as radiology, cardiology, laboratory, etc., the object-oriented view is of common services, e.g.: order entry, enterprise scheduling, results reporting,

etc. These services have many operations (methods) in common across the clinical departments. If they are created on an enterprise basis, they can be sub-classed to meet any detailed needs or nuances of specific clinical departments. A lot of duplicated functionality (in operations, staffing and software) could be eliminated with this approach. Technology users in Healthcare industry recognize the need for a standards-based, service-oriented approach to systems integration that enables continuous managed migration of computing technology. Architecture with a high level of common services across the system components, like the services to control and communicate professional activities, to identify patients, and to access knowledge secures a high degree of modular reusability.

The issues as described above are some of the issues facing enterprise systems and are addressed in this work. A system's architecture can serve as a "complete picture" for integrating the various subsystems and also serve to alleviate several of the above mentioned issues. In the following sections, we discuss how the notion of software architecture can be deployed for a framework in this context.

This thesis concentrates on one significant component in healthcare called 'Order Management'. This component is such that its functions span over all the different departments of healthcare and it interacts with several components within healthcare information system.

CHAPTER 4

ORDER MANAGEMENT

The current infrastructure has many monolithic systems with each system providing a number of functionalities. One such module of functionality is order management. Order management is required by a number of systems in various departments in a healthcare facility. We describe this component in this section and use this as a case study.

As described before, there has been no documentation for any of the components within healthcare. The only available document for Order Management was the User Manual for the current system in use. From Chapter 3, we gathered that Order Management by itself has not been developed as a separate component or system but it exists within another system (e.g. The STAR patient Care system). Hence, I had to study the use manual for the entire system and gather from it, the requirements of an order management component.

An order is a request for material or services, usually for specific patient. These services include medications from the pharmacy, clinical observations (e.g., vitals, I&O's) from the nursing service, tests in the laboratory, food from dietary, films from radiology, linens from housekeeping, supplies from central supply, an order to give a medication (as opposed to delivering it to the ward).

Orders are generated by physicians, and fed into the computer system by nurses or an application. Any person [or application] that places the order is called placer. A Healthcare facility is made up of different departments like Radiology, Laboratory etc. An order is placed for a test/ exam/ material from/ for a particular department. Once orders are placed they have to be communicated to the required department or filler (In HL7 terminology – a person or entity that carries out the order). This may involve interaction between two or more different systems. Hence, orders have to move between different systems, that could mean different environments in terms of information systems.

All clinical orders share a few common features:

- are for a patient.
- require some of the patient demographic information like ones name, sex, date of birth, medical record number (or identification number) etc.
- are always associated with some type of priority
- specify a requested date, time and/or shift
- are always issued by a physician.
- are also given some level of confidentiality (by the health care institution or the physician)

4.1 Significance of OM

An order is the central part of healthcare in terms of both business and clinical workflow. It not only triggers the clinical part of providing healthcare but is also responsible for initiating the financial workflow of the healthcare facility (HCF). Due to its characteristics, a system managing orders and its related functions has to interact with other systems handling registration (patient information), financial information and systems meant for specific departments. In other words an order management component ‘interfaces’ with all these other systems. But the current infrastructure does not recognize order management as an enterprise level component. It is always embedded within other systems and strongly coupled with other applications.

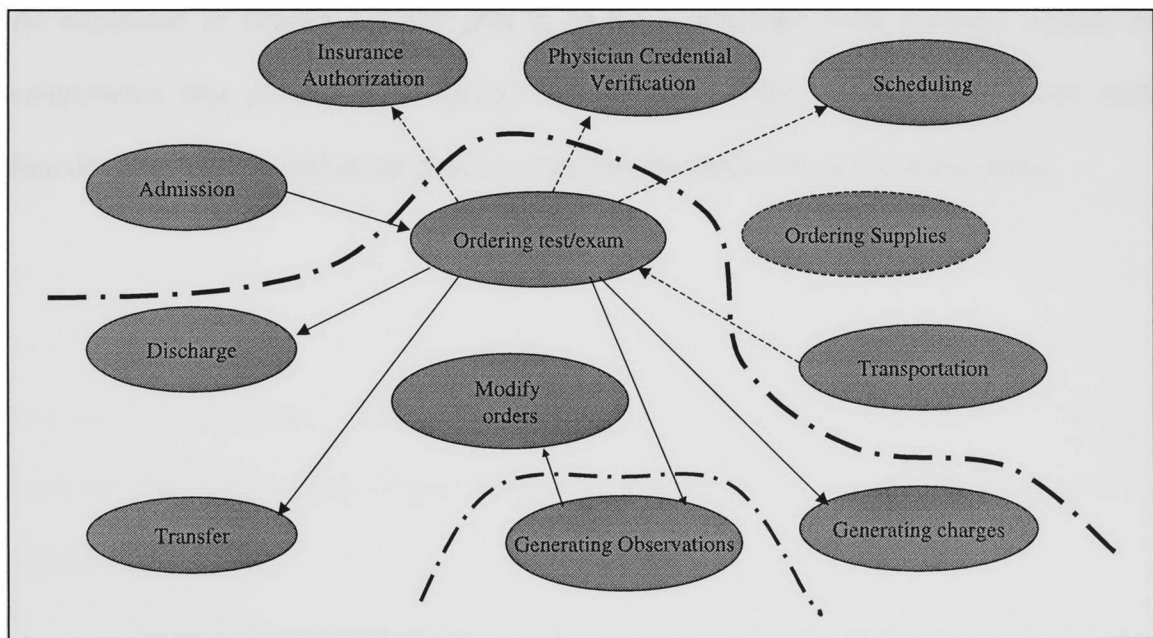


Figure 5 Business Processes Related to Order Management

Figure 4 indicates the business processes related to Order Management. The dotted arrows denote the dependency. The solid arrows indicate the order of flow of the business processes.

All the systems that have a need for this functionality implement the order management module. These systems are typically obtained from not one but various vendors. These systems need to interact with each other to achieve the business process of the organization. Hence system integration becomes an important and challenging issue and the healthcare organization, as an end-user has to invest in solving system integration issues and the maintenance of these integrated systems.

As explained in section 3.4, the goal is to have enterprise wide business objects or *components* that provide the required functionality. Order management is one such functionality and the following chapters describe the method to achieve this goal.

CHAPTER 5

PROBLEMS AND THEIR CAUSES

Our study of the enterprise in the healthcare industry revealed several distinct problems. Some of the problems addressed in this thesis are described below with respect to Order Management.

5.1 Problems in the Enterprise

The most common problems encountered during system integration are that of incompatibility between the components underlying hardware, operating system, programming language, and so forth. The current infrastructure uses the traditional point-to-point solution or the interface engine approach to solve these problems. Though this does achieve interoperability between systems in terms of sharing data, functionality is not shared between systems. This causes duplicated functionality and this makes the overall infrastructure inefficient. The end-users (HCF) end paying more for the duplicated functionality. The systems do not work together to achieve a common business goal because one system is not aware of the existence of another in terms of business functions.

There is often the problem of data mismatch even though interoperability is achieved through some means. The data that is communicated is often not explicit and causes

messages to be discarded if not properly understood. Though standard HL7 messages are used for communication, the way these messages are constructed can be made proprietary due to the flexibility that HL7 provides. This flexibility was provided to allow every HCF to add their necessary fields. But this very flexibility causes problems. There is also no easy method to track errors or loss of data. The only way this can be done is to check the log file that is created in the interface engine at the end of the day. This does not capture all errors like when a message is lost in the network, or when the receiving system discards a message.

There is too much dependency on the vendor from the user perspective. Any change required in the system in terms of interfaces is expensive and often requires too much time.

Since each of the systems in a Healthcare Facility (HCF) is proprietary, there is no enterprise knowledge of the functionality provided by the system. For example, the STAR patient care system contains the functionality for order management but this is not accessible to any other system that requires using this functionality. The functionality is not available across the enterprise and this also causes redundancy in functionality. We summarize the problems that are of prime importance to us.

Interoperability.

There are several means to achieve interoperability in terms of simple exchange of data without qualifying it or applying any type of semantics to it. But the existing

infrastructure does not achieve integration at a level of interoperability that ensures not only information sharing but also sharing of functionality and synchronization between components towards a common business objective.

Functional Redundancy.

This indicates replication of the same function in more than one component. This occurs due to the fact that systems are developed independent of each other for a particular need. For example, a healthcare organization procures an ancillary system like radiology information system (RIS) to handle radiology orders, which contains an order management module. But the central Patient care system also has an order management module existing within it that has the functionality to handle orders. Then the functionality to create orders and store order information is duplicated in subsystems, which together form one conceptual system. Organizations have to invest more money to obtain systems and pay for functions that they already have, since it comes so strongly coupled with the new functionality.

An end-user organization would obtain maximum benefit if it were possible to share data and functionality across different vendors' systems and integrate them to preserve their (the organization's) business logic and infrastructure.

5.2 Causes of these Problems

There are several reasons for the causes of these problems in the healthcare information system.

Lack of underlying infrastructure.

There is no underlying infrastructure for the healthcare systems. The systems have been acquired and integrated on ad hoc basis. Due to lack of an overall architecture the components within healthcare are not identified explicitly and this causes development of systems to be haphazard. During the development of a particular system there can be no assumption made about the infrastructure, hence the system incorporates all the required functionality. For example, the STAR patient care system and the RIS, both implement the order management component since the existence of the component in the enterprise infrastructure is not made explicit. This causes functional redundancy and high expenditure on the parts of the enterprise.

Lack of a common data model.

There is no common data model for the order management component that can state its functionality explicitly. Without a clear understanding of the component, integration of this component with other components is difficult. The subsystems within healthcare were not developed in an orthogonal fashion (they were not developed to be reused or integrated with other components). In general, it is very difficult to separate these systems into modules or change the way the modules work together. The end result is large monolithic systems with modules like order management embedded inside the system with no external access to its functionality.

Lack of component behavior structure.

There is no explicit definition or specification of the behavior of any of the components within healthcare. The order management component is embedded within monolithic systems and its behavior is neither well understood nor defined in a systematic way. Hence, even if other systems require use the functionality offered by order management, there is no explicit assumption about the component that they can make in order to use it.

Lack of common interaction pattern and common communication data model.

There is no explicit assumption about the nature of data communicated. Though most of the systems within healthcare use the standard HL7 messages to interoperate, these messages are not consistent across all implementations. This causes data mismatch when messages are communicated to systems that expect one type of format but get a completely different type of format. Since most of the systems were not built to interact with each other, there is no explicit definition of interaction patterns between communicating systems. This is another reason for ad hoc integration and custom solutions to achieving interoperability.

Lack of global architecture structure.

The Healthcare industry is far behind in the application of software architecture to their information systems. There is no concept of global architecture for an enterprise. Lack of such an architecture prevents the end-user from having a blue-print for system integration. Without architectural vision the order management component does not have a well-published interface that specifies the services required of the component. The

current architecture also never addresses the issue about how this component should interact with other components.

This work concentrates on the data model for order management seen as a component in the overall global architecture for healthcare. The requirement for the order management component is a well-defined component architecture and well defined interface architecture. We also outline future work that can be done to enhance this model to provide a more concrete solution to the problems stated above.

5.3 Our Approach

We are using an architecture-centered approach to address the complex and difficult problems in distributed healthcare system design, integration, and migration from legacy systems based on standard-based distributed object technology. The importance of software architecture in the development of systems has been emphasized earlier in chapter 2. Based on that and our requirements, we are using a service-centric architecture view that views the enterprise as a collection of services working together. It has been proven time and again, that systems with a good architecture (and with well-documented specifications) can withstand the vagaries of extension, modifications, and technology changes better than systems without one [CN96][2]. An architecture-driven approach will help us identify components in the system and their interdependencies. With proper abstraction at the analysis and design phase, a system may be built so that it is extensible, flexible and manageable, even when huge. We use OO concepts during the analysis and design phase and this brings with it the advantages of OO technology.

Systems in healthcare are typically distributed in nature. The requirements of an enterprise level system are that systems should be interoperable, flexible, legacy system compatible, and integrable. None of the existing solutions meet all the above requirements. Most systems from vendors are built independent of the other existing systems. Due to the sheer number of vendors, and in the absence of a universal standard or interoperability language, it is difficult to build functionality that lets one system talk to all others. Our solution is based on standard technology (CORBA) to build standard APIs. CORBA offers a standard interoperability language for systems with requirements such as above to be interoperable.

As mentioned in section 3.4, the current infrastructure in healthcare is that of vertical departmental systems that are bulky in terms of providing all required functionality within themselves. Our approach is aligned with the approach of the healthcare industry i.e. to have an object-oriented view of common services, e.g.: order entry, enterprise scheduling, results reporting, etc. These services have many operations (methods) in common across the clinical departments. If they are created on an enterprise basis, they can be sub-classed to meet any detailed needs or nuances of specific clinical departments. A lot of duplicated functionality (in operations, staffing and software) could be eliminated with this approach.

This type of architecture has several important implications if systems are implemented properly based on it. First, it can significantly simplify individual application

development. The common domain-specific services represents underlying concepts essential for conceptual integrity (in terms of service specifications), for vendors who want to develop an application (component), reducing development time and product costs. Second, it will significantly reduce the complexity of system integration and interoperation. This is because standardized interfaces and building blocks not only simplify integration between any two given applications, but also avoid potentially large number of customized mappings between the applications that could blow up exponentially (Figure 4). It significantly reduces the redundancy in functionality between different applications because common functions are implemented as shared services, and this also facilitates more effective software reuse.

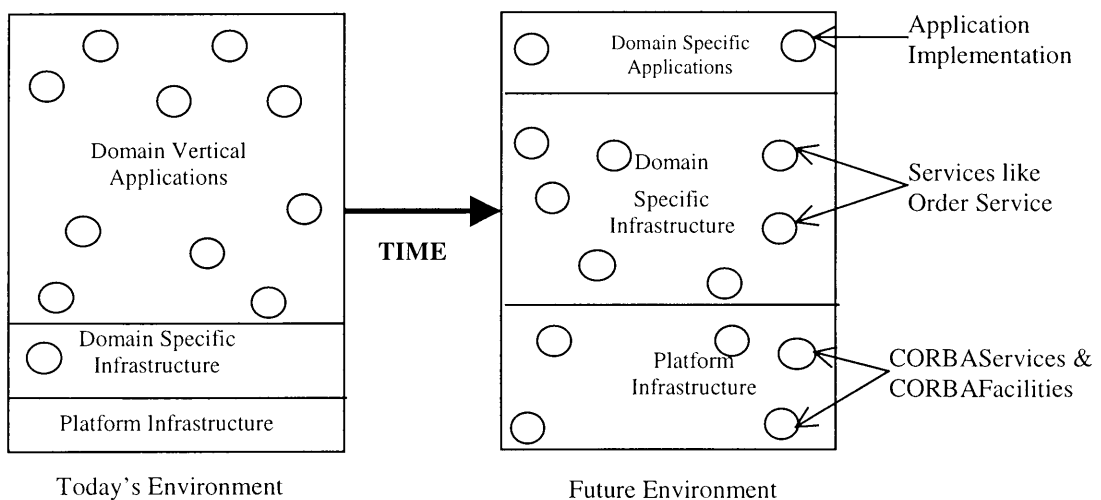


Figure 6 Moving toward standard-based architecture.

For example, such an architecture in the healthcare domain would define services for significant business processes like Patient Identification service, Clinical Observation Access Service, Lexicon Query Service and Order Service. The use of such an

architecture is further enhanced by building it based on a well-accepted architectural standard like CORBA. CORBA-based domain specific standard specifications leverage development through design reuse, commercially supplied software services, and interoperability. But since an enterprise system is comprised of numerous independent subsystems the complete set of requirements for the enterprise required to design the architecture for such a framework may not be fully understood at the start. Hence we take an incremental and iterative approach to develop this framework.

As new functions and components are identified and incorporated into the model, the service architecture will mature. The goal is to finally have an overall architectural framework for this domain industry (healthcare enterprise) such that it serves as a blueprint for enterprise level system integration, development of individual components, and component reuse. The following chapters describe such an approach with respect to the order management component.

CHAPTER 6

PROPOSED SOLUTION

Our solution is based on the outlined approach. This solution is use-case driven and is based on object-oriented analysis and design. Initial analysis helped understand the functionality of order management and workflow associated with orders.

6.1 The Order Management Business Workflow

This is a brief description of the order management business workflow. Without going into too many details at this point, this does give an overall view of what behavior is expected of this component and the environment that it is in. Once registration is complete and a patient is admitted to a healthcare facility, the patient is within the clinical context of the healthcare domain. Then a physician requests one or more orders for a patient. Then there is verification with the insurance company, which is usually done manually by a person. Once authorization is obtained from the insurance company, the order can be completed. To do so, the patient information obtained during registration is required. This is obtained from the registration system. The credential of the ordering physician is verified before the orders are placed within the system. Then the orders have to be sent to the respective departments for completion i.e. execution of the request. Once the orders are processed, financial charges are generated which lead to billing information. This cycle continues till the patient is issued a 'discharge' order, which is a

special order that indicates the end of that particular 'encounter'/episode of medical care. An encounter/episode is the period from an admission to a discharge for a particular patient when admitted in a HCF.

Our approach to tackle the problem in this domain has been to follow an object-oriented approach. In keeping with the approach, the first part of the analysis involved the use case analysis.

6.2 Use Case for Order Management

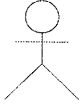
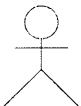
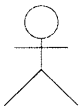
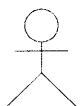
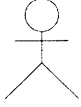
When a design is use case driven the whole system architecture will be controlled by what the users wish to do with the system. Having a use case diagram that reflects these requirements of the users, lets the modifications be reflected in the use case as a direct change in the user requirements. The use case model controls the formation of all other models and this facilitates traceability through all models.

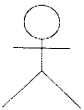
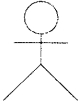
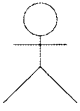
Figure 7 shows a Use Case for Order Management. This was derived from the Healthcare business domain after extensive interviews with several domain experts within BHS and is a reflection of the business workflow described in the earlier section. This final use case is a result of several iterations starting with a basic simplistic use case. The use case diagram identifies several use cases within order management. These use cases constitute a strong tool for defining the system functionality. They also act as support when developing subsequent models, since these models are based on the use cases.

The initial use case was created based on the primary use case i.e. 'Place Order'. This is the first use case that was identified. Eventually, through discussions, several more use cases became apparent. There had to be one to take care of generating charges, one that was responsible for delivering orders to the departments and one that would be used to modify orders. These were the first few that were obvious from the requirements and business case that I studied. As I generated use cases they were presented in front of experts and users from BHS. Based on their comments, I made further modifications to the use case model. We went through this cycle several times till we arrived at the current use case model. After one iteration, we found the need to add a separate use case for 'transfer' and 'discharge' order. Though these use cases are very similar to the 'Place Order' use case, they differ in the interaction with some actors. Hence, I created separate use case for them, which extends the behavior of the parent use case. The relationship between 'Place Order' use case and the actor 'Insurance' was at first designed as 'authorize'. But after some discussion with some experts, it was concluded that *all* orders need *not* be authorized by the Insurance e.g. a discharge order. Hence the relationship has been changed to '*may* authorize'. The names of the use cases have been modified after some suggestions to reflect the view from outside i.e. from an actor's viewpoint.

Following is the description of the final use case that was designed as a result of several iterations and discussions. The description is based on a *Use Case* Template borrowed from a *Use Case Template* draft [Col98].

Table 1 Description of Actors from the Use Case

Actor	Description
 Placer	A person or system who places an order into the information system. An order once placed could generate further orders and thus take the role of a placer.
 Filler	A person or system that executes the order. This could be technicians, physicians or systems that are deployed to perform certain tests or exams.
 Registration	A business process during which most of the encounter management related actions are triggered and encounter information is captured. The component[s] providing this information to the order management component in order to place an order, is termed Registration in the above Use Case.
 Housekeeping	A system [department] that keeps track of the housekeeping facilities within the healthcare facility like beds and rooms to be allocated etc. Some orders that affect the housekeeping need to be communicated to the department.
 Financial System	A system that keeps track of the billing information and the financial charges incurred by a patient, when an order is placed or processed.

 Scheduling System	A system responsible for scheduling the execution or processing of orders (in this case) by indicating a date/ time for the orders to be processed and allocating a resource for the order to be processed.
 Transportation Department	A system that deals with transporting facilities like a stretcher, a wheelchair etc. and is responsible for assigning a facility to a patient as and when required. This could be required for any order.
 Insurance	A person or system that acts as a contact with the insurance company. This is required to authorize the execution of certain orders.

Use Case 1. *Place Order*

Description *Placer* creates an order into the system after the physician has placed an order.

Assumptions none

Actors Placer

Insurance

Steps

1. Placer receives order to be placed (from the physician). Placer may be a designated nurse, a technician, a person specifically assigned for that purpose or in some cases the physician himself.
2. Placer starts the process to place the order (Refer to Sub-Use Case 1.1).
3. The order may be authorized by the Insurance Company.
4. The details of the order are entered into the system.

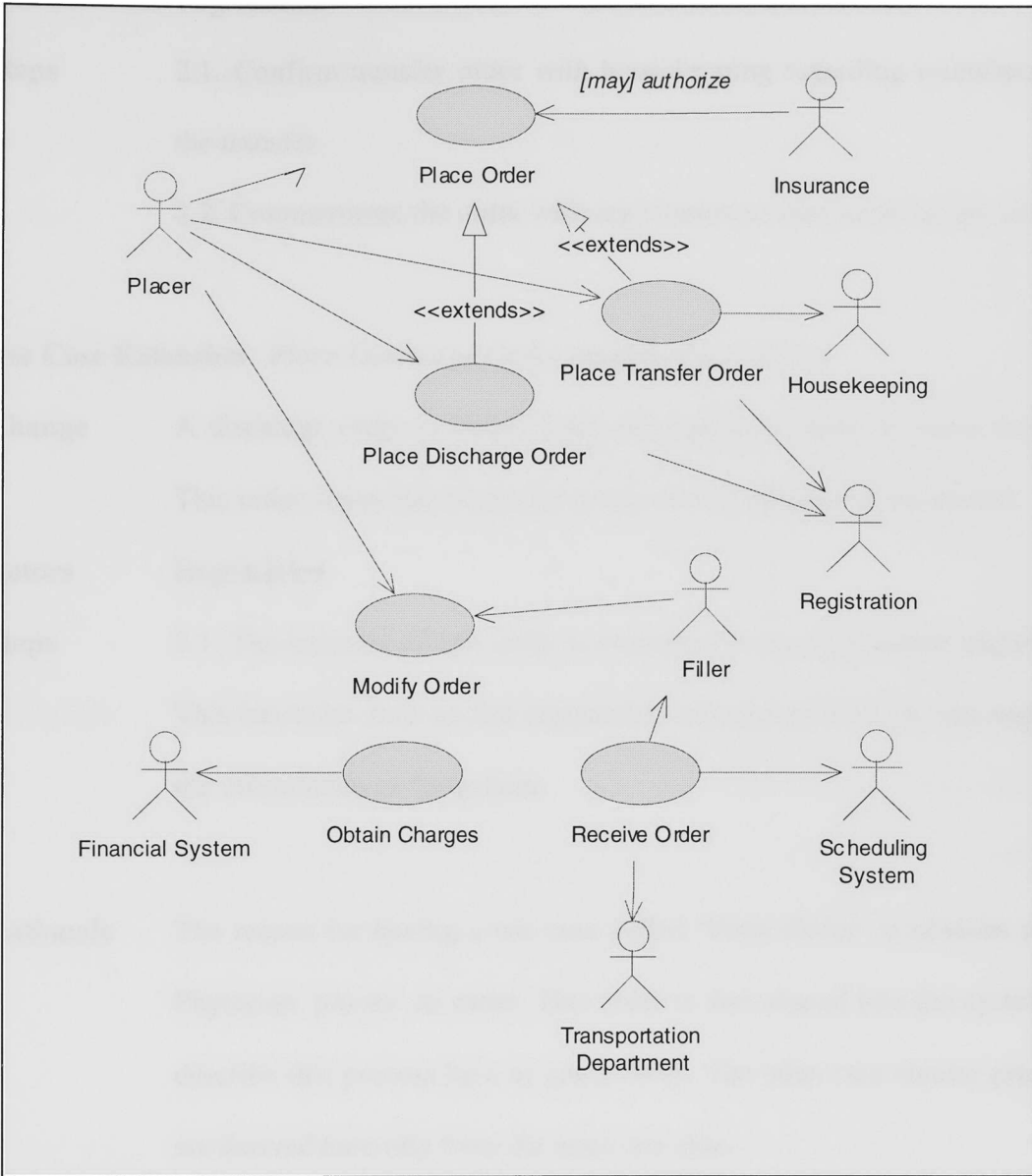


Figure 7 Use Case for Order Management

Use Case Extension *Place transfer order extends Place Order*

Change A transfer order is a special order that is placed for the transfer of a patient admitted into the Healthcare Facility (HCF), from one room to another.

Actors Housekeeping

Registration

- Steps**
- 2.1. Confirm transfer order with housekeeping regarding room/space for the transfer.
 - 2.2. Communicate the order with registration so they know of the change.

Use Case Extension *Place Discharge Order extends Place Order*

Change A discharge order is issued when the patient is ready to leave the HCF.
This order closes the encounter or the current episode of the patient.

Actors Registration

- Steps**
- 2.1. The issuance of this order is conveyed to the registration department.
This has to be done so that registration can update their records regarding the information of the patient.

Rationale The reason for having a use case called 'Place Order' is obvious since a Physician '*places*' an order. This order is then placed into the system. We describe this process here as place order. The other two similar processes are derived naturally from the main use case.

Use Case 2. *Receive Order*

Description The order that is placed into the system is communicated to the various actors interacting the environment.

Assumptions none

Actors Scheduling System

Filler

Transportation Department

Steps

1. Order is scheduled through the scheduling system.
2. IF transportation required THEN 2.1 Notify Transportation Department
2.2 Confirm availability of transportation
3. Order is delivered to the Filler (the person and/or system that executes the order).

Issues

none

Rationale

Every order has to be sent to a particular department or 'filler' who would then complete or execute the order. The order is thus 'received' by the 'filler'. Hence the other departments or components receive the order from the order management component. This can be achieved in one of two ways: either the order management component sends the orders or the other component requests the orders. Either way, the other components receive it. Hence, we require a use case that describes this process.

Use Case

3. *Modify Order*

Description

An order may be modified at any point in time. The modifications could be changes made specifically to the order such as change in the date and time requested for the order. Or the order could be cancelled if there is found to be a lack of need for it. The status of the order changes when it is

scheduled and after it is executed. This has to be indicated with respect to the order.

Assumptions none

Actors Placer

Filler

Steps

1. The order to be modified is identified.
2. The modification is recorded.
3. The information regarding the person/system making the modifications is noted.

Rationale Every process goes through stages of modifications. And order also goes through several states. Certain values are attributed to the order during this modification and this has to be recorded. This gives rise to a separate activity in which an order is modified and the parameters associated with it are recorded. Hence, we have a use case called '*Modify Order*'.

Use Case 4. *Obtain Charges*

Description After the order is placed or after it is processed (depending on the requirements of the business), charges are generated for every order that was placed. These charges are then fed into the financial system where it is processed.

Assumptions none

Actors	Financial system
Steps	<ol style="list-style-type: none"> 1. Charges generated for each order. 2. Generated charges communicated with the financial department.
Issues	When are the charges exactly generated? After the order is placed but before it is executed? Or after it is executed?
Rationale	As orders generate a clinical workflow, they also give rise to a financial workflow by generating charges. Every order is associated with a charge or the cost of performing the test/exam. This is generated once the order is placed into the system or once it is executed depending upon the policies of the Healthcare Facility (HCF). Either way, the financial system or component needs to access these charges. Hence a use case called ' <i>Obtain Charges</i> '.

As we can see from this use case, the Order Management component interacts with various other components (Systems) like Registration, Housekeeping, Financial system, Scheduling system, transportation system etc. The following table gives a description of each of these actors.

This use case model aims to define the limitations of the component and specify its behavior. This gives an explicit definition of the behavior of the system that will also be reflected in the other models for order management. This information can be used by other systems during system integration to ensure correct behavior.

The 'Place Order' use case was the first use case that was identified. The use case was then identified to be composed of several sub-use cases or activities. These are specific activities that have to take place before an order can actually be placed in the system. These activities have been considered significant enough to entail a sub-use case diagram to depict the dependency. The following figure (Figure 8) shows the Place Order use case and identifies the activities involved in placing an order.

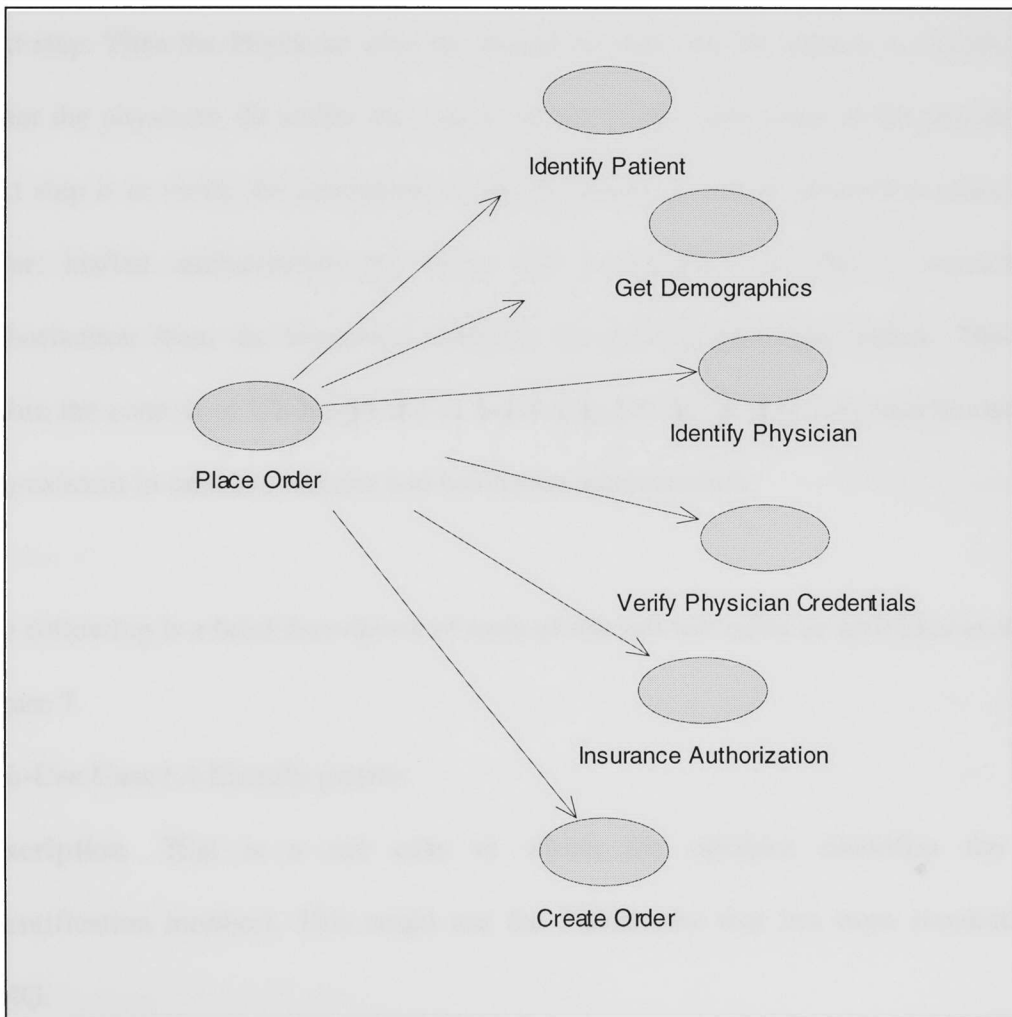


Figure 8 Sub-Use Case for Place Order

Place order use case is composed of several activities that can be described by a sub-use case as shown in the figure. The arrow drawn from **Use Case Place Order** to the sub-use cases denotes a '*include*' relationship.

To complete the process of placing the order, first the patient has to be identified in the context of the healthcare facility. This could be done using his name, his patient identification number (or any ID that is used in the HCF), or the floor (if he is admitted into the HCF). Once the patient is identified, getting his demographic information is the next step. Then the Physician who has placed an order for the patient is identified using either the physician ID within the context of the HCF or the name of the physician. The next step is to verify the credentials of the physician: is he/she allowed to place such an order, his/her authorization etc. Once this is clarified, all that is needed is the authorization from the Insurance company for placing particular orders. This is true within the context of US healthcare in most cases though it may vary and be completely non-existent in certain countries and healthcare infrastructure.

The following is a brief description of each of the sub-use cases or activities as shown in Figure 7.

Sub-Use Case 1.1 Identify patient.

Description This is a use case in which the operator identifies the patient (Identification number). This could use the PIDService that has been standardized by OMG.

Use Case Referenced By Use Case 1. Place Order

Steps 1. A patient is identified using his/her unique identification number.

2. The search can also be done by other parameters related to the patient is if the unique identification number is not known or does not exist for some reason.

Sub-Use Case 1.2. Get Demographics

Description In this process, the related demographic information (name, sex, etc.) is retrieved from the relevant system. This could use the proposed Demographic service.

Use Case Referenced By Use Case 1. Place Order

Steps 1. The demographic information for the identified patient is retrieved.

Sub-Use Case 1.3. Identify Physician.

Description The operator is required to identify the physician who is responsible for generating the order. This generates a relationship between a physician (ordering physician) and the order that was generated. We may need a service that manages this relationship.

Use Case Referenced By Use Case 1. Place Order

Steps 1. Identify Physician from a list of physicians using the physician identification number; or any other parameter designated by the Healthcare Facility like the last name of the physician.

Sub-Use Case 1.4. Verify Physician Credentials.

Description Once the physician is identified, his credentials need to be verified. This includes verifying whether the physician is authorized to give that particular order.

Use Case Referenced By Use Case 1. Place Order

Steps 1.

Sub-Use Case 15. Insurance Authorization.

Description Before the order can be processed or even placed in the system, there has to be authorization from the insurance provider. This is also done by the operator though there may be some orders which do not need this verification (e.g. Discharge order).

Use Case Referenced By Use Case 1. Place Order

Steps

1. Call the insurance company.
2. Obtain authorization for specific orders.

Sub-Use Case 1. Create order.

Description This is where the operator actually enters the data specific to the order like the test/ exam for which the order has been made, the priority of the test, the date and time requested, and other information specific to the type of order.

Use Case Referenced By Use Case 1. Place Order

Steps

1. Create an order for a particular patient with a unique identification number.
2. Enter all the specific data to the Order into the system.

This use case diagram for Order Management gives explicit description of the architectural context of the component. It describes the interaction of the component (Order management) with the other components or subsystems of the Healthcare

Enterprise wide system. This facilitates understanding of the context of Order Management and aids in system development. It also helps define the 'boundary' that separates the 'external' components that interact with orders from the information that resides within the component.

6.3 The Information Viewpoint of Order Management

Figure 9 is the RM-ODP² information viewpoint model for the Order management component in the healthcare information framework.

The data model that is detailed below consists of entity objects or classes identified from the use cases discussed in the earlier section. Some of the entity objects are obvious while others are not. The entire model went through several iterations similar to the use case model. The initial version looked like the following figure (Figure 9). As the figure shows, it has an object representation of a person, who might have one or more encounters, where each encounter would have one or more orders. An order is either a simple order for a test/exam or could be of composite nature where it contains several orders within itself. This has been modeled using the composite pattern [8- pg. 163]. Furthermore a simple order can be of two types, one-time (that occurs only once) and recurring (occurs repeatedly at fixed time intervals). This has been modeled as subclasses of Atomic orders. Since an order can be in several states, there was a need to represent the state of an order, hence the object representation 'ObjectState'. This model was presented in front of the experts from Baptist Health Systems (BHS) and found to be

² RM-ODP stands for The Reference Model for Open Distributed Processing [xi]

lacking in several important features. The final model is shown in Figure 10, and the description of the model in detail follows that.

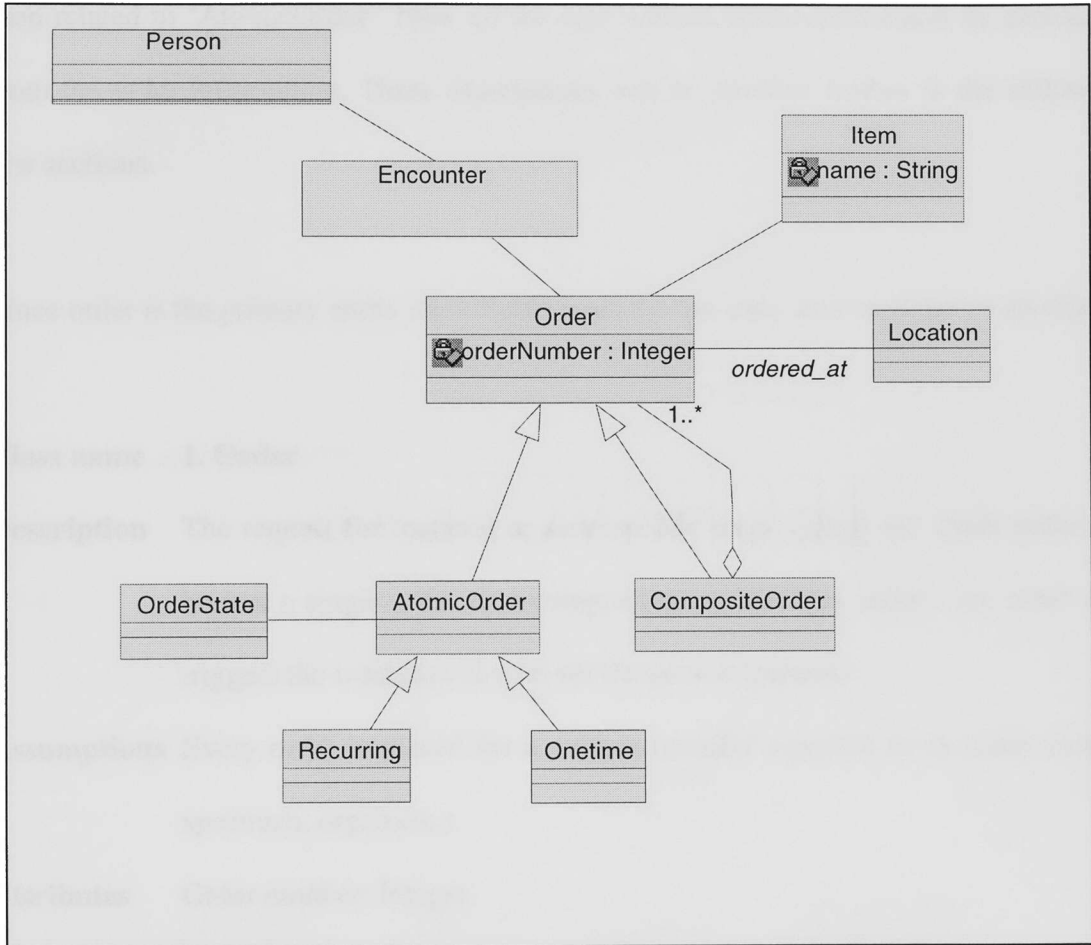


Figure 9 Initial Data Model for Order Management

Deriving from the comments by the experts there was found to be the need for representing the person responsible for placing the order, as well as the person changing the state of the order. Furthermore, the two object representations for 'Person' and 'Encounter' were collapsed into one, now called 'SubjectOfCare'. This was done because

orders can be placed not only on persons but also on specimen, which have no relationship with Orders. Also the notion of encounters is not global, so it may or may not exist. Instead of directly sub-classing atomic orders, a better way to design the typing of orders is to create a new class called 'Type' and then sub-class this class. This class is then related to 'AtomicOrder'. Now all the type related information could be abstracted from the order information. These descriptions will be clarified further in the following few sections.

Since order is the primary entity identifiable from the use case, it is modeled as an object.

Class name 1. Order

Description The request for material or services like tests, exams etc. Each order can be just a single order or a composition of multiple orders. An order also triggers the workflow to care for the person (patient).

Assumptions Every order is placed for a subject (usually a person or in some cases a specimen, organ etc.).

Attributes Order number: Integer

Variations *AtomicOrder*: A single order that performs one and only one request for some material or a service for a patient like a single test, e.g. a chest x-ray.
CompositeOrder: A Composition of multiple orders that contains more than one single order - it could be a composition of more than one *AtomicOrder* or *CompositeOrder*.

AtomicOrder and *CompositeOrder* inherit from *Order*, all the characteristics that are typical of an order. *CompositeOrder* exists for the sake of convenience in the medical realm. These are orders that seem like a single order but actually disintegrate into a number of *AtomicOrders*.

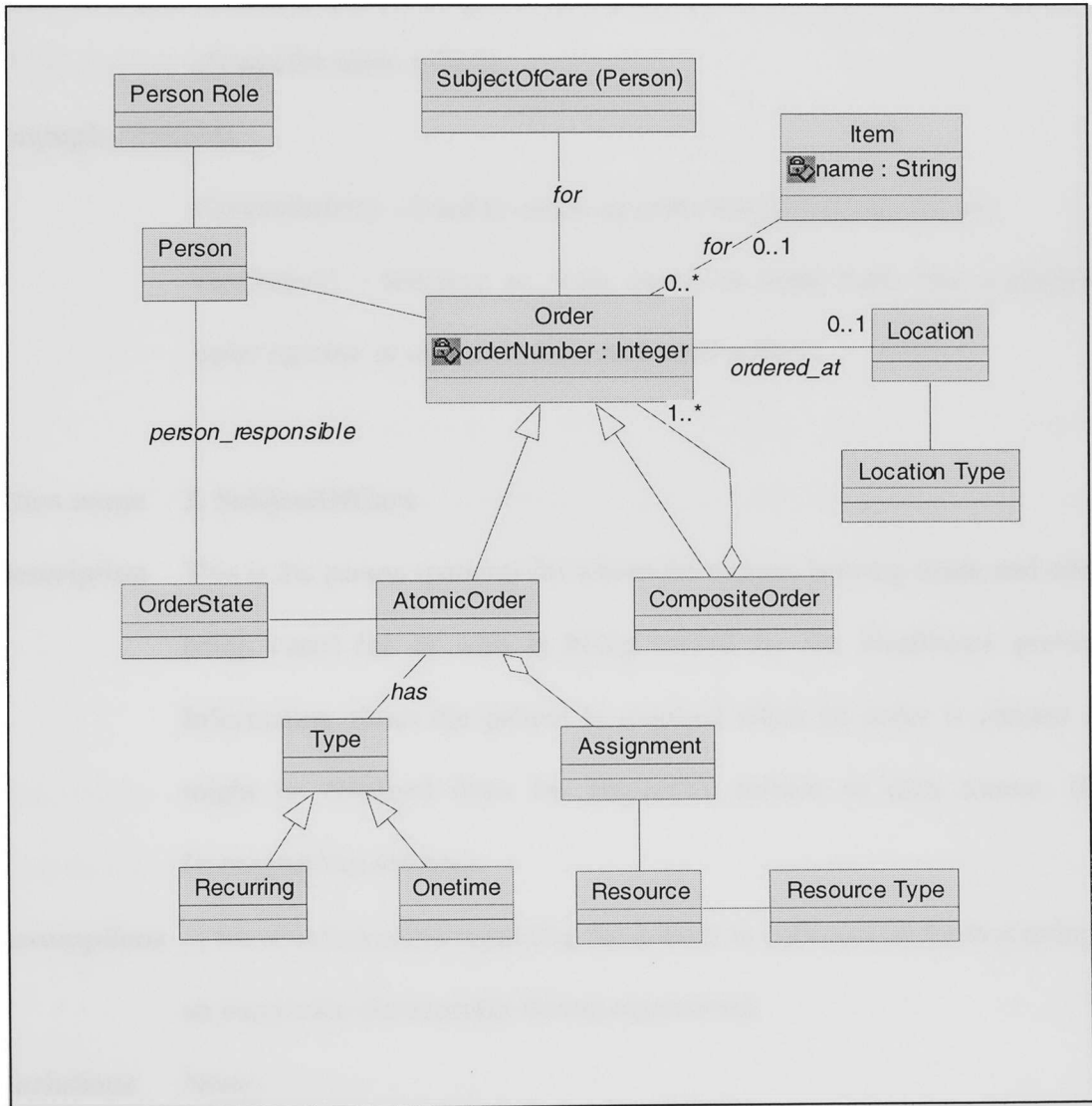


Figure 10 Conceptual view of Order Management

Rationale An order as described earlier is the primary entity in the order management component. An order can be atomic or composite. And since they are only types of orders, they are designed as 'is-a' relationship.

Traceability Use Case '*Place Order*', '*Modify Order*'.

Dependencies *SubjectOfCare* – requires this class to be associated, since orders are always for some subject.

Example Methods

CreateOrder() – Used to create an order with given parameters.

GetOrder() – Retrieve an order based on some logic like a particular order number or all orders for a particular patient.

Class name **2. SubjectOfCare**

Description This is the person (patient) for whom the request is being made and who is being cared for or who is being served by the Healthcare provider. Information about the patient is required when an order is created and might be obtained from the respective service or data source. (E.g. Demographics service).

Assumptions A lot of information regarding the patient is collected at the beginning of an encounter, for example during registration.

Variations None.

Rationale The need to capture the information about the subject for whom the order is being placed gives rise to this class. It is designed as a separate class

since the information it carries comes from another component. So it is not tightly bound to this particular component.

Traceability Use Case '*Place Order*', Sub-use case '*Identify Patient*' and '*Get Demographics*'. These use cases would require a class called SubjectOfCare.

Dependencies It provides support for Class Order.

Example Methods

GetSubjectName() – Returns the name of the subject.

GetSubjectDescription() – Could be used to find out some kind of description of the subject, for example, inpatient or outpatient.

GetSubjectParameters() – Where depending on the parameters the function could be changed i.e. a function could be defined to return the patient's address based on his identification number. Different logic could be used to return different parameters.

Class name 3.Person

Description This is the person responsible for the order in different ways such as physician (who is responsible for generating the order), nurse/ clerk (who is responsible for placing the order into the system and also for making any changes to the order) etc. Information about the physician who makes the request (authorizes the creation of the order) might be obtained from a different data source or service. (E.g. PHiDS – Physician Identification

service). This person can exist in different roles, which is qualified by *Person Role*.

Assumptions Information about any such person associated with the system is always available.

Variations None.

Rationale There is a need to capture the information of the person responsible for a number of activities (use cases) within order management like the person responsible for placing the order, the person updating the status of the order etc.

Traceability Use Case '*Place Order*', Sub-use case '*Identify Physician*'. Use Case '*Modify Order*', '*Receive Order*'

Dependencies *Person Role* – requires this to describe the role played by a particular instance of the class

Example Methods

GetAllRoles() – Used to find all the roles a particular person had with respect to the system.

GetPersonRole() – Could be used to find a particular role that a person is playing at a given instance.

Class name **4. Person role**

Description Identifies the role of the person instantiated through Class Person. This class has an '*association*' relationship with class Person.

Rationale Since the role of the person could vary depending on the context, we have a class called '*Role*', which would let the HCF decide which roles are important enough to be recorded.

Traceability Use Case '*Place Order*', Sub-use case '*Identify Physician*'. Use Case '*Modify Order*', '*Receive Order*'

Dependencies Provides a role for a person.

Example Methods

CreateRole() – Create a specific type of role for a person.

Class name 5. Item

Description This is the actual test/ exam/ material for which the request is being made. This could be maintained as a separate data source (database).

Assumptions None.

Variations None.

Rationale Every order is associated with a type of test or exam and each of these have a fixed charge associated with it. Maintaining this as a separate class gives the flexibility to implement this in different ways such as having a separate database for it or having the charge value incorporated in the class.

Traceability Use case '*Obtain Charges*' and '*Place order*'

Dependencies None

Example Methods

CreateItem() – Could be used to create an item.

GetItemPrice() – Could be used to get the price of a test/ exam.

Class name 6. Location

Description Every order is placed at some location. There might be a need to record this depending on the business needs of the healthcare provider (HCP). There can different locations that might be important enough to be recorded, depending on the context.

Assumptions None.

Variations None.

Rationale The location may or may not be recorded depending on the needs as defined by the HCF. Defining this a separate class gives the flexibility to either use it or not.

Traceability Use Case '*Place Order*', '*Modify Order*' and '*Receive Order*'

Dependencies *LocationType* – Required this to define the type of location that is specified in this class.

Example Methods

GetLocationType() – Get the location type for a particular location

Class name 7. Location type

Description The need of recording different types of locations necessitates another class called *LocationType*, which qualifies the type of the location. So if the location that is to be recorded is an office of a physician, the *Location*

object will contain the actual location of the office while the *LocationType* object will describe the type of location as being 'office of physician'.

Assumptions None.

Variations None.

Rationale LocationType can be used to describe the type of location.

Traceability Use Case '*Place Order*', '*Modify Order*' and '*Receive Order*'

Dependencies Provides type for class *Location*.

Example Methods

CreateLocationType() – Used to create a new location type.

Class name 8. OrderState

Description The status of the order during the workflow of providing care to a person keeps changing. And it is required to record the status of the order at every stage. Some of the different statuses that an order can have are 'Requested', 'Scheduled', and 'Completed'. Depending again on the needs of the HCP, *OrderState* can be sub-classed to define more statuses.

Assumptions None.

Variations None.

Rationale As described in the use case, every order will go through different states. This has to be recorded along with associated parameters. *OrderState* is used to record the state along with certain parameters. *OrderState* can be further sub-classed to get certain types of states which require specific parameters to be recorded.

Traceability Use Case '*Modify Order*'.

Dependencies Provides state for class *Order*. Associated with class *Person* responsible for changing the state.

Example Methods

ModifyState() – change the state of a particular order.

Class name 8. **Type**

Description Every order has a type that can be either a one-time order or recurring. Recurring order indicates an order that repeats after a certain time interval; this could be several hours or several days. A one-time order is executed only once.

Assumptions None.

Variations Type is sub-classed into One-time and Recurring orders.

Rationale Since orders can be of different types, there has to be provision for taking care of it. We consider only atomic order to be either recurring or one-time. After several discussions with clinical experts it was concluded that composite orders did not recur as a whole. Either individual order within the Composite order recurred or did not. Hence, we have *Type* as a *association* relationship with *AtomicOrder* and then have sub-classed it to give two more classes.

Traceability Use Case '*Place Order*'.

Dependencies Required by class *Order* to identify the type of order.

Example Methods

CreateType() – Create the type of order.

GetRecurInterval() – this could be used for type recurring order to find out the period of recurrence for the order.

Class name 9. Resource

Description For an order to be executed, it might require the use of certain resources like an X-ray room, certain modalities like a cat-scan machine or other specialty technicians (who for the purpose of this model are considered resources). This class captures resource information, which can be used to schedule a particular order.

Assumptions None.

Variations None.

Rationale Resources are definitely required during clinical workflow.

Traceability Use Case ‘Place Order’ and ‘Receive Order’.

Dependencies *Resource Type* – requires this in order to classify its type.

Example Methods

GetResourceType() – Returns the type of the resource.

RequestResource() – ask for a particular resource to be assigned for an order.

ResourceGranted() – When a resource is granted this can be used to convey the schedule for the resource.

ListResources() – Can be used to list all the resources.

Class name 10. Resource Type

Description Since each resource can be of different types as described above (like rooms, modalities or technicians), a class is used to qualify the resource.

Assumptions None.

Variations None.

Rationale To describe the resource and its type we define two different classes, once again for flexibility.

Traceability Use Case '*Place Order*' and '*Receive Order*'.

Dependencies Provides type for class *Resource*.

Example Methods

CreateResourceType() – Creates a resource type.

Class name 11. Assignment

Description These resources have to be assigned for a particular order and have to be scheduled to do so. During this frame the orders have some association with resources. This class captures the relationship between orders and resources when a resource is scheduled for a particular order.

Assumptions None.

Variations None.

Rationale This is derived from a design pattern called '*Business Patterns of Association Objects*' [9]. This relationship is captured as a separate object so that information related to the association between orders and resources can be recorded. For example, an order may be placed and this order may

require a particular resource. At that time the association between the order and the resource is that of requirement. Once the order is scheduled and a resource is assigned to the order the relationship changes.

Traceability Use Case '*Place Order*' and '*Receive Order*'.

Dependencies Acts as an association between class *AtomicOrder* and class *Resource*.

Example Methods

GetScheduleInformation() – Used to get the information associated with the resource when it is scheduled for an order

This is an intuitive idea of a conceptual view of order management obtained after extensive in-depth study of the domain of healthcare using the expertise at Baptist Health Systems of South Florida (BHSSF). The business information for order management is captured in this data model. The data model is a generic framework, which can be (re-)used for the development of this component.

The work in developing the data model entails distributing the behavior specified in the use case descriptions among the objects in the data model. An object can be common to several different use cases. Any change in the user and/or business requirements is reflected in the use case and this can easily be mapped to data model. The data model can also be used to develop the interface diagram that will help generate the interface specification. The interface specification for a component providing such information should define the interaction between systems [components] that wish to communicate

order information. The specification must also state the functionality supported or provided, thus making it explicit.

CHAPTER 7

AN ENTERPRISE VIEW

In the last chapter, we looked 'into' Order Management. In this chapter, we look at Order management with respect to other components in the overall enterprise and Order Service as part of the service architecture for healthcare.

7.1 Solution: A 'Good' Architecture

A good architecture means well-defined modules, well-defined interfaces and uniform interaction patterns. If a system is developed with well-defined modules and well-defined interfaces, other components have explicit assumptions to deal with. A component with well-defined modules will have only the relevant functionality and information encapsulated within itself. This will ensure that functionality is not duplicated in different components within the same large-scale system. Uniform interaction patterns will ensure that the behavior of the component is consistent. Without a well-defined architecture, order information resides in different systems (due to redundancy in functionality) and information often becomes inconsistent. Other systems/ components do not have access to the required order information and very often do not know how to obtain it.

The problem is that of integrating existing legacy components into new and novel configurations. A large-scale system, like healthcare will always follow an incremental approach to development. It is indeed difficult, if not impossible to construct the overall

architecture of the system without identifying the individual components of the system. Identifying and developing each component is the first step in achieving an architecture centric approach to system development. Order management has been identified as one essential component of a healthcare information system that offers enterprise wide functionality. It has been shown how this component can be developed efficiently using object-oriented methods and an architecture has been developed for it. This architecture can be further complimented by the following work, which is to be developed via CORBAmed efforts.

The use cases and the data model described in Chapter 6 provide the groundwork to further develop an Order Service based on CORBA but domain specific to healthcare. This service would be part of the service-based global architecture for the healthcare domain. The following sections detail on how such a service can be built and fit into the overall architecture.

7.2 Order Service

Order Service would define the set of interface specifications in CORBA IDL and would specify the functionality provided and the interaction between all components that need or provide order information. Clinical Order Service is a mechanism for co-ordination of data about clinical orders between information repositories and different applications. This service should also provide for write access to the clinical information. A system may use the Order Service to send clinical order information to other systems or a system

that needs to share order information may notify other applications/ systems about it via Order Service.

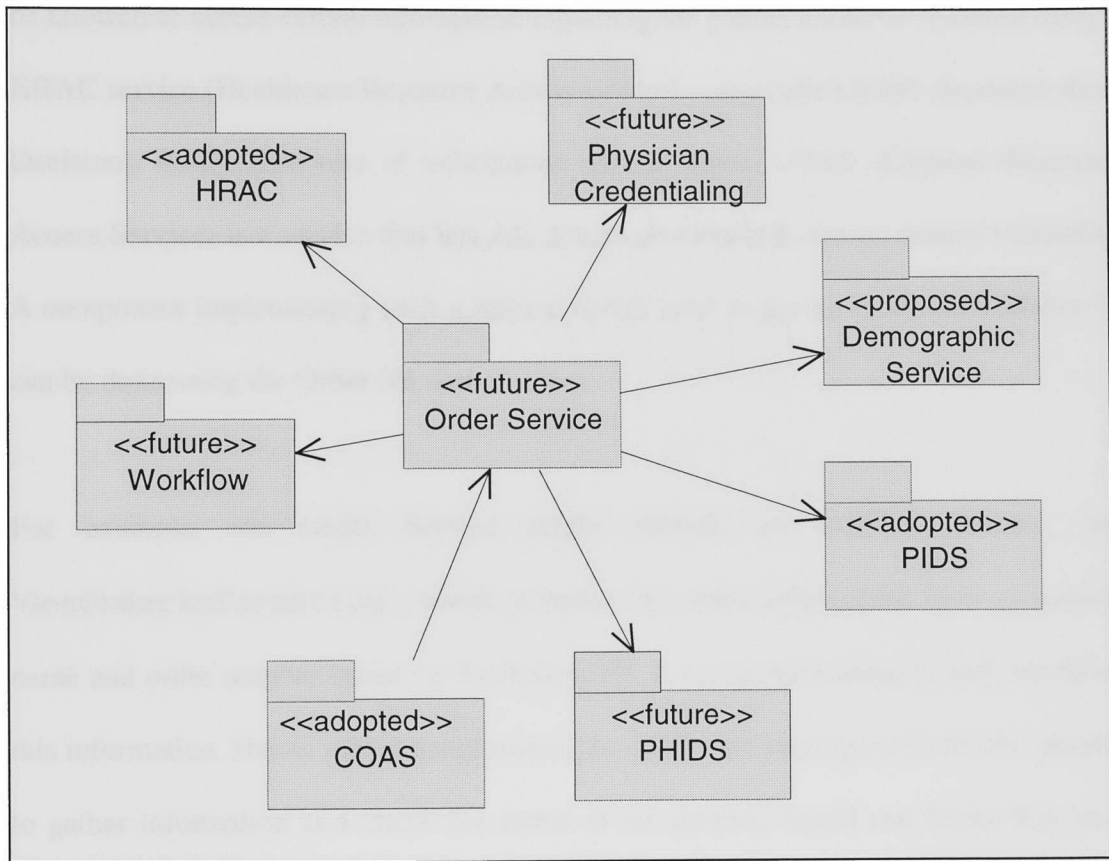


Figure 11 Dependency diagram for Order Service

All such dependencies would be described in the architectural description of the order service as shown in figure 10. Order management component depends on several other systems for information; hence order service would depend on other services that offer such information. One such service would be the proposed Demographic service which would let one access demographic information regarding a patient. The Order management component would then use the services offered by the demographic service

to get the required information. Similarly, the component could use services offered by a service like PIDS (Patient Identification Service) to lookup the patient based on his identification number across more than one domain. The security issues like who would be allowed to access certain information regarding the patient could be resolved using the HRAC service (Healthcare Resource Access Control - now called RAD, Resource Access Decision). RAD takes care of information access issues. COAS (Clinical Observation Access Service) is a service that lets one access observations and its related information. A component implementing such a service would need to access order information. This can be done using the Order Service.

For example, the Order Service might provide an interface service called 'GetOrderInformation', which provides the order information such as test/exam name and order number based on the Patient ID. A component using COAS would need this information. Hence such a component (like a software package built for the physician to gather information and check the status of his patient) would use Order Service via COAS. Similarly, a component requiring the use of Order Service to place an order will have to use services offered by RAD, PIDS, Demographic Service. Other foreseen services include PhiDS (Physician Identification Service) which could be used to identify physicians based on certain parameters and Physician Credentialing Service, which would be used to verify physician authorization and credentials. Order Service is seen to be dependent on these services. Workflow service and its relationship to Order Service is explained in further detail in Appendix B.

The following sequence diagram (figure 11) shows an instance of where an Order Service would fit into the workflow of the healthcare business process. This is a specific instance of placing a radiology order and then its execution. The example has been considerably simplified for ease of understanding.

1. First a Radiology order is placed i.e. created by an operator using the Order Service.
2. Then the order is scheduled by a scheduling system. The scheduling system uses the Order Service to obtain the orders.
3. The Information system at the Radiology (RIS) receives the scheduled orders via the Order Service. Order Service could use CORBA service such as Event Notification Service to notify such peripheral systems of scheduled orders.
4. The ancillary system notifies the Order Management system through the Order Service about the updated status of the order.

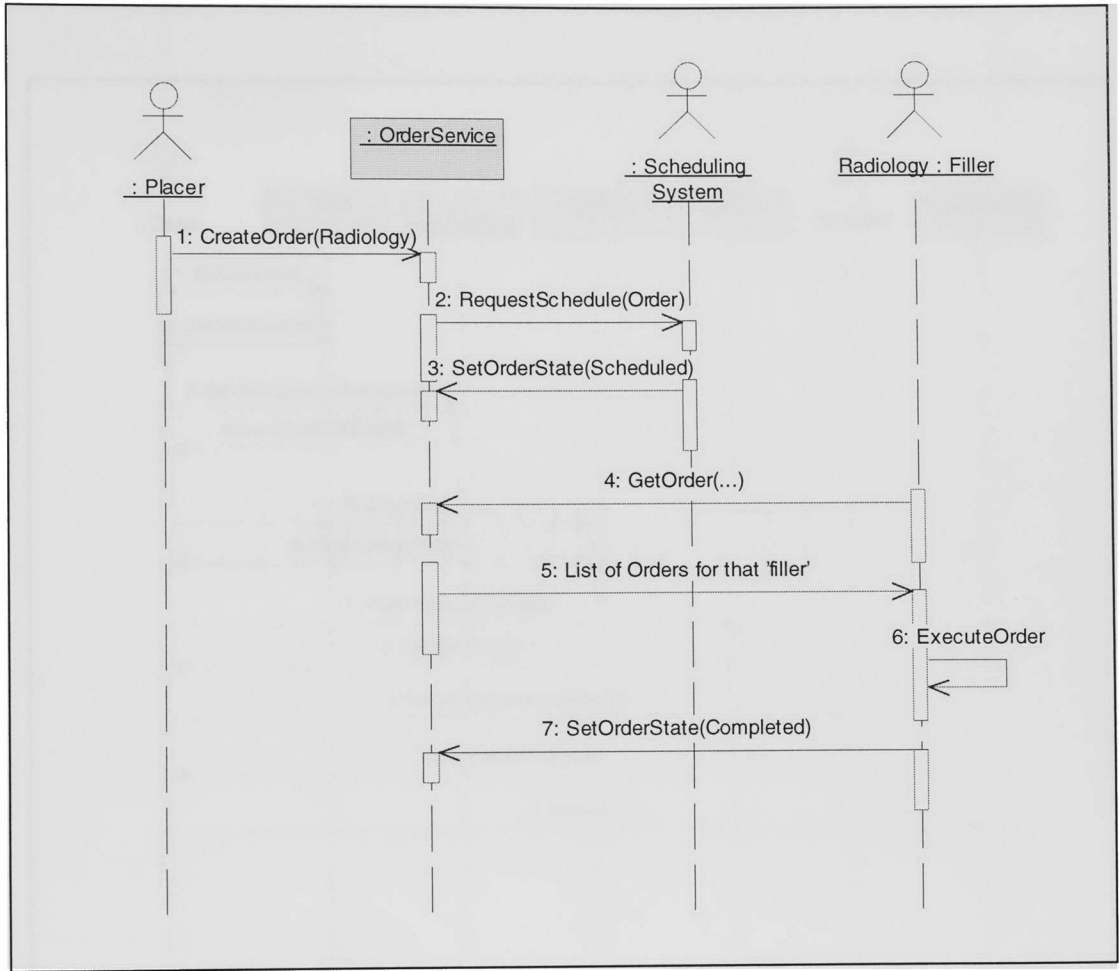


Figure 12 Sequence Diagram for processing a Radiology Order

There are more such instances since Order Management system interacts with several other critical business systems and all of them share order information. The following figures show all the other interactions.

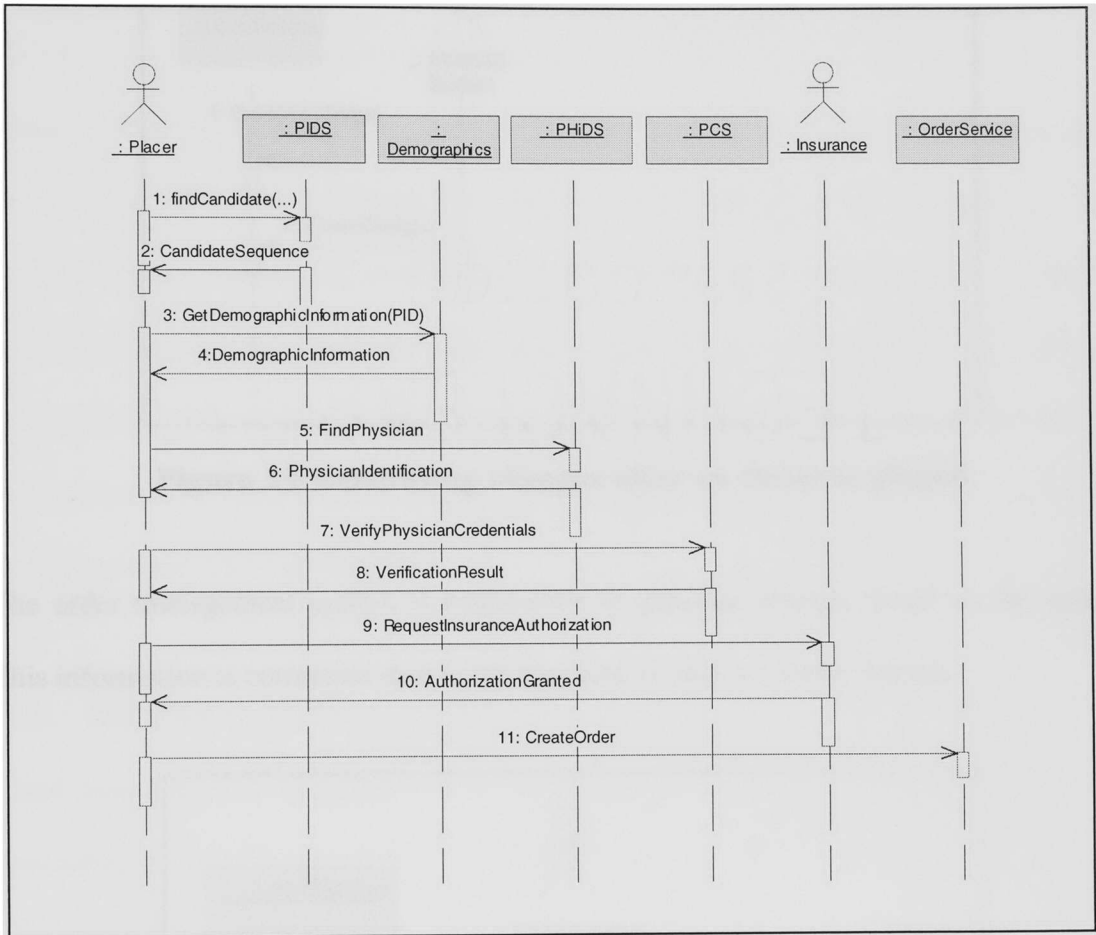


Figure 13 Create an order

This is the possible sequence to place an order using an Order Service. PHiDS and PCS are services to identify physicians and verify physician credentials respectively. These services are not proposed yet.

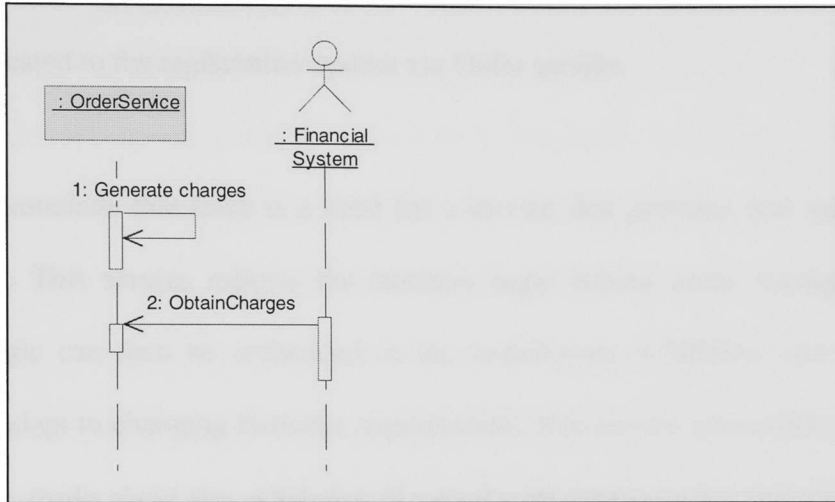


Figure 14 Generating charges after an Order is placed

The order management system is responsible to generate charges based on the orders. This information is communicated to the financial system via Order Service.

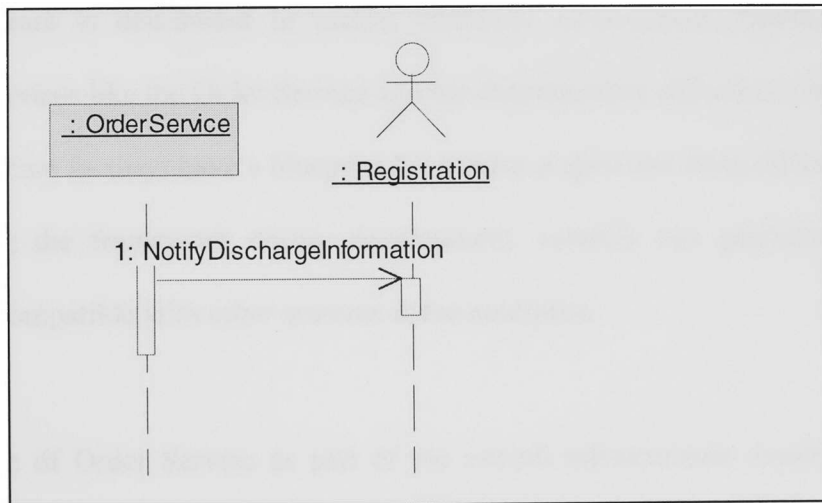


Figure 15 Notification after patient discharge

The following diagram is the interaction of order service with the registration system.

An encounter comes to an end with the discharge order for the patient. This information is communicated to the registration system via Order service.

Hence, we conclude that there is a need for a service that provides and manages order information. This service reflects the business logic behind order management. This business logic can then be embedded in the middleware (CORBA) and this enables systems to adapt to changing business requirements. The service also explicitly states the assumptions made about the semantics of component functionality and interfaces. The use cases and the data model developed earlier should be used as the basis to develop the interface model for Order Service.

7.3 Architecture for an Order Service

Since healthcare is distributed in nature, existence of a service based architecture describing services like the Order Service and the dependencies will ensure that end-users (like a healthcare facility) have a blueprint for system acquisition from different vendors. By following the framework during development, vendors can guarantee that their systems are compatible with other systems in the enterprise.

The existence of Order Service as part of the overall infrastructure would ensure that vendors developing systems that need to use the functionality provided by this service need not include it as part of their system. This not only saves development time and cost but maintains conceptual integrity of the overall system.

Architecture specifies the dependencies between this service and other healthcare domain services. Thus, once specified, this will add to the overall 'horizontal' framework across the enterprise and help refine it with more details. Figure 10, as described above, shows the dependency relationship between the Order Service and the other services that are (or might be) part of the overall healthcare service architecture.

This infrastructure acts as a skeletal structure for the healthcare domain. A healthcare organization that needs a Healthcare Information System can derive from this basic architecture, the components that are specific to its needs, and then start development or deployment of available systems that were also developed conforming to this infrastructure. Since this domain-specific architecture is based on CORBA, it inherits all the advantages that CORBA provides for distributed object computing. It provides the infrastructure for integrating components with legacy systems and common interoperability language for communication between systems.

7.4 Summary

We summarize the justification of this framework with respect to the issues discussed earlier.

Interoperability

The CORBA infrastructure provides basic interoperability: the platform and common language for communication of information. The framework for healthcare based on CORBA provides the business functionality and logic required for healthcare. By

standardizing an Order Service, integration of components could be simplified. Each of the components requiring the use of this functionality would have to conform to the standard interface and this could be achieved at the time of development of these components. Since integration is simplified, the healthcare organization can now afford to acquire components that best suit a particular need like a Radiology system developed using the latest technology. The use case diagram helps understand and describe the functionality of the order management component explicitly.

Redundancy

Order information could continue to reside in the different systems. The CORBA Notification Service could be used within the Order Service to notify other systems when data in one system changes. This is one of the ways that inconsistencies in data can be avoided.

The overall framework for healthcare will ensure that there is reduced functional redundancy as systems develop over time. Since, the framework would define a clean division of functionality; the components would be developed without duplicating other functionalities. The Order Service provides a set of interfaces for one such functionality. The functionality of this component is clearly indicated by the generic data model that was designed.

The above work contributes a step towards achieving an overall framework for healthcare. There have been numerous attempts at defining architecture for large-scale

systems. But there has been no significant effort at studying or researching and enterprise level systems where numerous heterogeneous systems interact in the scale as described earlier. This work contributes partially towards such an effort. By accumulating such pieces of work over time a generic architecture for the healthcare enterprise can be developed. This architecture would then be used for system development, system acquisition and system integration.

The core part of this work will be used to enhance the CORBAmed Healthcare System Template (CHST) being developed by CORBAmed as the guideline for healthcare information systems [vi]. An Order Service would fit into the 'Provider Centered Services' within a CHST. As defined in the CHST document, these services are chiefly aimed at provider support. This package is seen to provide the following services:

- Creation of a plan of clinical and administrative activities (or events) that may be executed by the care plan execution service
- Invokes the establishment, request, performance and other states of work items
- Utilizes information about planned activities to develop worklists for specific agents

The descriptions of these functions partially match the description of an order service. Hence, an order service would definitely contribute to the structure of the template.

But what is different about this design as compared to the traditional or existing methods of solving the same problem? The object-oriented method of analysis and design is definitely more systematic and organized way to approach a problem. Traditional methods have always relied on incremental but ad hoc development. With organized

design where dependency is traceable, development become more methodical and maintenance in turn is simplified to a large extent. One noticeable element that has been missing in development efforts thus far has been good documentation. With the writing of this thesis serving as a base, further development for the healthcare framework will have documentation to guide the process. The methodology of arriving at the solution is organized in such a way that solution seems clearer.

As addressed before, having a standard-based service infrastructure (described by CHST) will also make development and system deployment easier. The dependency of the components is better defined when this approach is followed. The use case diagram and the data model give a clear picture to any developer who wants to either start implementing the system from scratch or for a person who is looking to deploy a system that uses the order management component.

One of the existing ways to integrate systems and/or components is CORBA. But what this architectural design adds to it is a framework for healthcare systems. Healthcare systems can be built and integrated using CORBA and it will definitely work. But to have an overall picture of the healthcare system and to be able to see the dependency between is an added asset to the advantages of CORBA. This thesis is a contribution to the development of such a framework.

CHAPTER 8

CONCLUSION

It is clear that enterprises are moving toward open, standard based architecture and there is a need to build next generation information system. Though architecture models like OMA/ CORBA prove to be extremely beneficial in solving several architectural issues (like certain types of low-level mismatches), providing specific extensions for domains could further enhance the benefits. For complex systems within particular domain, the architecture can be extended with domain specific standards and specifications. These specifications are for Services that define business objects that can be applied horizontally across the domain. The idea is to shift from department-based vertical systems to 'horizontal' services that can be used across the domain.

Using the Healthcare domain as an example, we realize that identifying key components within the domain is a starting point in developing such an architectural framework. We identify one such component, which can be used as a horizontal service across the healthcare domain and propose defining standard specifications for it.

Identifying such components within the entire enterprise will eventually help develop an infrastructure that can be used and reused to build domain specific large-scale systems. It could also serve as a guideline for long term enterprise development.

Future issues/ potential

There is much effort required to build an overall framework for the healthcare domain similar to architectures that exist for other domains like Telecommunications [TINA]. We recognized one approach to doing this as being based on CORBA. This thesis has concentrated on providing information and an object framework for one business component of healthcare. Further work is needed to define the service and the set of interfaces and standardize them. This would probably be the next issue on the agenda of the OMG meeting of CORBAmed. During the course of the work, a relationship was also realized between Order Management and Act Management, which is the European Union effort towards healthcare information systems [Appendix A]. This relationship needs to be investigated further. Orders are central to every workflow that occurs within the clinical context. So is there a direct relation between Orders and workflow? Yes, this definitely seems to be the case. There is already a Workflow Management Facility within OMG drafted by the Workflow Management Coalition group (WfMC). Applying this to healthcare domain seems natural and beneficial. So is there a need for a separate service for Order management or should it part of Workflow Service? This is being investigated currently in a separate piece of work [Appendix B]. And this issue would be brought up in front of the WfMC and CORBAmed, with the help of BHS.

Other efforts required in this effort include a methodology to validate this model. Since the whole idea of applying software architecture to large-scale systems is in its infant stage, certain methods or tools to help validate this idea would be beneficial. Recording every piece of information in the form of a repository that can be accessed by every IT

professional, not only in Healthcare but in other domains like Telecommunications, and Defense would be useful to the IT industry. Such an incentive is being carried out by the Interoperability Clearinghouse consortium [1].

BIBLIOGRAPHY

Web References

- [i] <http://www.cpri.org> - Computer-based Patient Record Institute page contains high-level description of the core element (CPR) and functionality (CPR system) for the CPR environment.
- [ii] <http://www.mcis.duke.edu/standards/guide.htm> - Health Informatics Standards page. Contains links to other useful health informatics sites.
- [iii] <http://www.mcis.duke.edu/standards/HL7/hl7.htm> - Health Level-7 standards page. History of HL-7, mission, and other HL-7 related information.
- [iv] <http://www.snomed.org/sdm/sdm.htm> - DICOM (The Digital Imaging and Communications in Medicine) related information.
- [v] <http://www.mcis.duke.edu/standards/CCOW/index.html> - The Clinical Context Object Workgroup Web page.
- [vi] <http://www.omg.org/homepages/corbamed/Roadmap/roadmap.htm> - The CORBAmed roadmap page. It gives the business case for healthcare information systems, the place of CORBA in healthcare and the progress of CORBAmed.
- [vii] <http://www.omg.org> - The Object Management Group's home page.
- [viii] <http://www.omg.org/corbamed> - The CORBAmed home page i.e. the Medical/Healthcare domain Task Force's home page.
- [ix] <http://www.sei.cmu.edu> - The Software Engineering Institute (SEI) at Carnegie Mellon University (CMU).
- [x] <http://source.asset.com/stars/lm-tds/Papers/ReusePapers.html> - Papers describing the relationship between architecture, domain-specific architecture and reuse.

- [xi] http://www.dstc.edu.au/AU/research_news/odp/ref_model/ref_model.html -
Description of RM-ODP and related concepts.

Books

- [1] Brooks, F.: *The Mythical Man-Month – Anniversary Edition*, Addison-Wesley, 1995.
- [2] Bass, L., Clements, P., and Kazman, R.: *Software Architecture in Practice*, Addison-Wesley, 1997.
- [3] Shaw, M. and Garlan, D.: *Software Architecture: Perspectives on an emerging discipline*, Prentice Hall, 1995.
- [4] Mowbray, T. and Zahavi, R.: *The Essential CORBA*, John Wiley & Sons, 1995.
- [5] Rechtin, E. and Maier, M.: *The Art of Systems Architecting*, CRC Press, 1997.
- [6] Edited by Andy Carmichael, *Developing Business Objects*, Cambridge University Press, 1998.
- [7] Allen, P. and Frost, S., *Component-Based Development for Enterprise Systems*, Cambridge University Press, 1998.
- [8] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.
- [9] Edited by Martin, R., Riehle, D. and Buschmann, F.: *Pattern Languages of Program Design 3*, Addison-Wesley, 1998.
- [10] Mowbray, T. and Malveau, R., *CORBA Design Patterns*, John Wiley & Sons, 1997.

Papers and Publications

- [GAO95] Garlan, D., Allen, R. and Ockerbloom, J.: Architectural Mismatch or Why it's hard to build systems out of existing parts, *Proceedings of the Seventeenth International Conference on Software Engineering*, 1995.
- [GS94] Garlan, D. and Shaw, M.: An Introduction to Software Architecture, CMU SEI Technical report, CMU/SEI-94-TR-21, ESC-TR-94-21, 1994.
- [GP95] Garlan, D. and Perry, D.: Introduction to the Special Issue on Software Architecture, *IEEE Transactions on Software Engineering*, April 1995.
- [GAO94] Garlan, D., Allen, R. and Ockerbloom, J.: Exploiting Style in Architectural Design Environments, *Proceedings of the ACM SIGSOFT '94 Symposium on Foundations of Software Engineering*, New Orleans, December 1994.
- [GP94] Garlan, D. and Perry, D.: Software Architecture: Practice, Potential and Pitfalls, Panel Introduction, *16th International Conference on Software Engineering*, Sorrento IT, May 1994.
- [AAG93] Abowd, G., Allen, R. and Garlan, D.: Using Style to Understand Descriptions of Software Architecture, *Proceedings of ACM SIGSOFT '93 Symposium on Foundations of Software Engineering*, Software Engineering Notes, pg. 9 – 20, December 1993.
- [Sha96] Shaw, M.: Truth vs. Knowledge: The difference Between What a Component Does and What We Know It Does, *Proceedings of 8th International Workshop on Software Specification and Design*, March 1996.

- [PW92] Perry, D. and Wolf, A.: Foundations for the Study of Software Architecture, *Software Engineering Notes*, ACM SIGSOFT, vol. 17 no. 4, pg. 40 – 52, 1992.
- [KLBK] Kazman, R., Lung, C., Bot, S. and Kalaichelvan, K.: An approach to Software Architecture Analysis for Evolution and Reusability.
- [Cle94] Clements, P.: From Domain Models to Architectures, *Workshop on software Architecture*, USC Center for Software Engineering, LA, 1994.
- [Cle95] Clements, P.: Understanding Architectural Influences and Decisions in Large-System Projects, *First International Workshop on Architectures for Software Systems*, Seattle, April 1995.
- [WC197] Wolf, A., Compare, D. and Inverardi, P.: Uncovering Architectural Mismatch in Component Behavior, University of Colorado, Dept. of CS Technical Report, CU-CS-828-97, Feb. 1997.
- [Par72] Parnas, D.: On the criteria to be used in decomposing systems into modules, *Communications of the ACM*, vol. 15, pg. 1053-1058, December 1972.
- [ABH93] Abowd, G., Bass, L., Howard, L. and Northrop, L.: Structural modeling: An Application Framework and Development Process for Flight Simulators, CMU SEI Technical Report, CMU/SEI-93-TR-14 ESC-TR-93-192, Aug. 1993.
- [KC97] Kazman, R. and Carrière, J.: [Playing Detective:] Reconstructing Software Architecture from Available Evidence, CMU SEI Technical Report, CMU/SEI-97-TR-010, 1997.

- [BBG99] Baum, L., Becker, M., Geyer, L., Molter, G.: A Process View on Architecture-Based Software Development, *Proceedings of first Working IFIP Conference on Software Architecture (WICSA-1)*, San Antonio, Feb. 1999.
- [BBG98] Baum, L., Becker, M., Geyer, L., Molter, G.: The Role of Architecture for Complex Systems Development, *Proceedings of 11th international Conference on Software engineering and its applications*, Paris, France, Dec. 1998.
- [BBM98] Baum, L., Becker, M., Molter, G., and Geyer, L: Using Software Architecture as a Catalyst for Reuse, *Proceedings of European Reuse Workshop*, Madrid, Spain, Nov. 1998.
- [CN96] Clements, P. and Northrop, L.: Software Architecture: An Executive Overview, CMU SEI Technical Report, CMU/SEI-96-TR-003 ESC-TR-96-003, Feb. 1996.
- [Pri90] Prieto-Diaz, R.: Domain Analysis: An Introduction, *Software Engineering Notes* 15, 2 (April 1990): 47-54.
- [BNW98] Butler, E., Navarro, E., and Wreder, K.: “*HealthCare that really delivers!*”, *Distributed Computing*, Vol. 1, Issue 9, September 1998.
- [Cza97] Czarnecki, K.: Leveraging Reuse through Domain-Specific Software Architectures, *Eighth Annual Workshop on Institutionalizing Software Reuse (WISR)*, Ohio State University, March 23-26, 1997.
- [Col98] Coleman, D.: A Use Case Template: draft for discussion, *Hewlett-Packard Software Initiative*, June 19, 1998.

APPENDIX A

ACT MANAGEMENT [RICHE PROJECT]

Act Management was first used in the ESPRIT RICHE project in the development of an open hospital systems architecture [NF95]. It is being used in several projects in the European Union and the act management model has been incorporated into Version 2 of the UK NHS's Common Basic Specification (CBS).

Act management comprises two main components: act state and the requester-performer relationship. An act is a provision of care. The definition of an act covers any activity provided in a hospital, for the benefit of a patient, requested by a "Requester" and performed by a "Performer". This definition is illustrated in the following Entity-Relationship model:

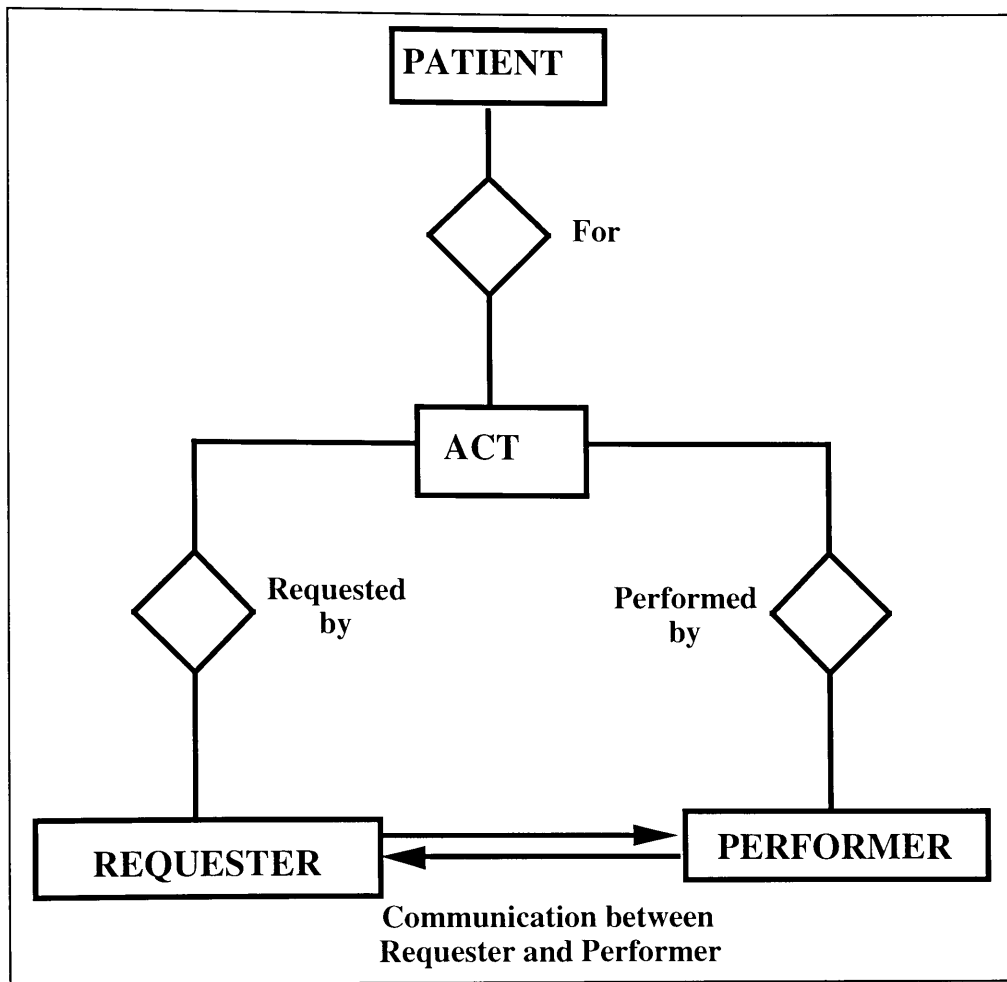


Figure A - i Acts as basic elements of the requester-performer relationship

This definition covers the acts requested by a physician in a clinical ward and performed in a medical support unit as well as "technical nursing acts" requested by a physician and performed by a nurse in the same unit. An act may change the patient status or the knowledge about his status, for example:

Consultation

Surgical operation and associated anaesthesia

Visit

Drug administration

Lab test and associated sampling

Meals

X-ray examination

Bandages

Collection of vital signs

Admission to a care unit

This definition of Act is very similar to the definition of **order** within US healthcare, except that it covers more than an order like admission to a care unit.

Elementary acts and protocols

An elementary act represents an activity at the lowest level. Its performance requires a set of resources such as time, staff members, location and equipment. It uses consumable objects such as drugs or small equipment (needles, tubes, etc.). A protocol is a cluster of elementary acts answering to a well-known medical need and is composed of elementary acts or protocols, such as Coloscopy or Glucose tolerance test, etc.

This is the same as **composite** and **atomic orders** where a composite order is a cluster of atomic orders.

The act life cycle

An act can be followed throughout its life cycle. It changes its status according to different events such as:

- The request of an act for a patient, by a "Requester" (physician, nurse, etc.) to a "Performer" (physician of a specialist consultation, medical support unit, the requester himself, nurse, etc.),

- the acceptance of the requested act, its suspension or its refusal by the performer (for technical reasons for example),
- the scheduling of acts, usually by the performer, by the requester himself, or by a special third party (e.g. central reservation office),
- the performance of the act by the performer that provides uninterpreted results which are available to the requester,
- the validation/interpretation of the results by the performer which is used to produce the report.

An **order** goes through exactly the same life cycle and changes its status accordingly.

Class of acts and Actual acts

In the Act Manager, distinction is made between the class of act (or act type) and the actual act performed for the benefit of a patient. The class of act provides the theoretical and pragmatic description of the provision of health care. The description of the classes of act and their environment corresponds to the "knowledge" information as opposed to the daily activities information attached to the patients and to the acts.

The Requester-Performer Relationship

Act management makes a distinction between requesters and performers of acts and establishes a relationship between them. In some cases, the requester and the performer may be the same person, but the distinction still applies because the requester-performer

relationship defines roles, not individuals. Requesters and performers co-operate to perform the act at each step of the act *life cycle*. This is illustrated in Figure A-1.

Act Management is part of the RICHE reference architecture for Hospital Information Systems, which was developed from the recognition that Europe requires an open approach to healthcare information systems to overcome the language and cultural barriers that prevent the European market being larger than it is. An open framework allows hospitals to mix and match their system components from competing suppliers who nonetheless are collaborating on the definition of the architectural standards. This architecture is well accepted in Europe and it has received good response from the user group as well.

It would be beneficial to study the relationship of Acts and orders and the application of the RICHE reference architecture with respect to the US healthcare industry. Is **'order'** a sub-type of an **'Act'**? Act Management is very similar to **'Order Management'** and this needs to be investigated further. This is being done under the concept of 'Task Management' discussed in Appendix B.

[NF95] Nicklin, P. and Frandji, B.: Act Management and Clinical Guidelines, *Health Telematics for Clinical Guidelines and Protocols*, IOS Press, 1995

APPENDIX B

TASK MANAGEMENT FOR HEALTHCARE

There is an attempt being made by the CORBAmed DTF to assess its requirements against the work of OMG Workflow group and the work on Act Management which has been going on in the European Union [Appendix A]. CORBAmed has adopted the term "task management" to cover this area in order to differentiate itself from other, similar, work. Tasks with respect to the healthcare means an activity provided in a healthcare facility, for the benefit of a patient.

Workflow

Workflow is concerned with the automation of procedures where information and tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal³. Workflow Management Facility specification introduces a workflow framework and interfaces to enable workflow aware applications and different workflow products to interoperate and work together. This specification is based on the reference model and architecture developed by a large group of workflow vendors and users under the umbrella of Workflow Management Coalition (WfMC). It is intended that such specifications will enable interoperability between heterogeneous workflow products and improve integration of workflow applications with other services, thereby improving the opportunities for effective use of workflow technology within the IT market, to the benefit of both vendors and users of such technology.

³ Reference: OMG Document Number - bom/98-06-07, *Workflow Management Facility*

The following is a brief description of the WfRM as described by the WfMC:

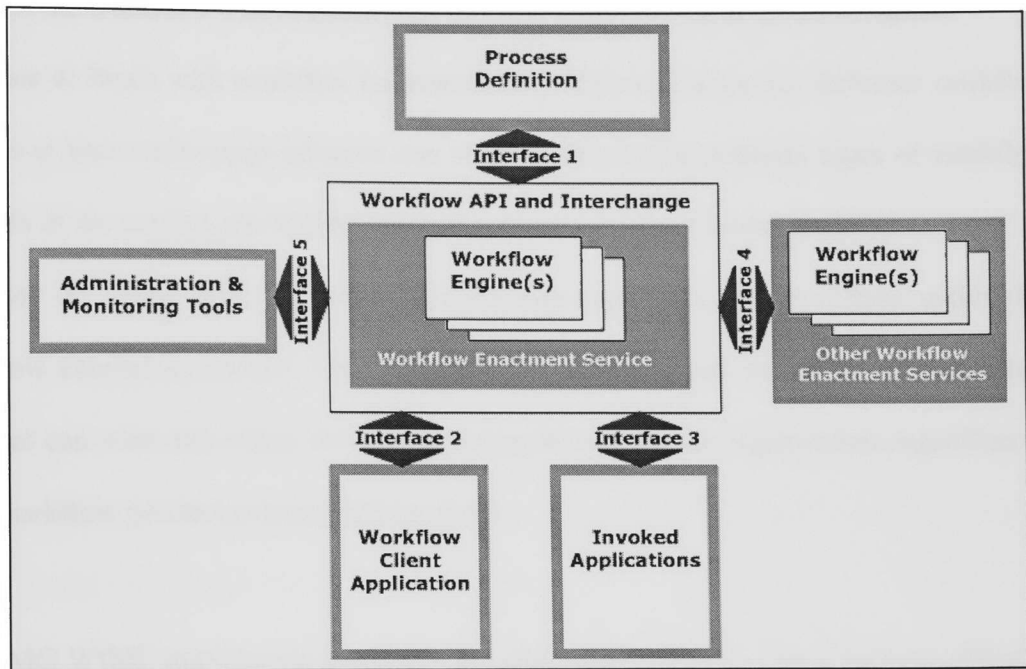


Figure B - i Diagram of WfRM

Interface 1: Deals with establishing an interface to process definition tools (CASE tools, UML, etc.). This allows business process modeling to occur within specialized tools and for workflow aspects of those business process models (BPM) to be defined into a process definition which can be passed into a Workflow Enactment Service (WfMF).

Interface 2: Deals with the passing of workflow information to client applications for manual tasks. The interface would allow the workflow enactment service to pass tasks to the client application for further human intervention. (i.e. When a task is completed, send message to client for next action)

Interface 3: Deals with the workflow enactment service invoking outside applications to finish any tasks/processes/activities that are part of the workflow (i.e. Once a report is finished, the workflow enactment service invokes a fax service to send the report).

Interface 4: Deals with workflow interoperability. This will allow for different workflow vendors to pass work items between one another. Due to the different types of workflow products in the market, the WfMC foresees different levels of interoperability.

Interface 5: Deals with the administration and monitoring of workflow within the workflow enactment service. This will allow for a centralized workflow administration tool that can view the status of work flowing throughout an organization regardless of what workflow product is being administered.

The OMG WfMF implements Interfaces 2,3 and 4 and part of 5. The following indicates the applicability of each of these interfaces in the healthcare domain.

Interface 1 deals with the process definition or protocol, i.e. to allow process definitions for the workflow aspects of the BPM's to be passed into the service. A method is required to define (dynamically or customized to the HCP) process definitions (protocols or clinical pathways). These protocols indicate a fixed path of care for certain ailment. For example, there is a fixed protocol to deal with hip replacement called "Primary" Total Hip CareMap™ within BHS. This defines all the orders that are to be placed for a patient with this ailment and its timeline. The idea is to automate this into a protocol that can then be implemented by the Workflow engine in the Workflow management system.

Interface 2 would deal with the user (filler), that would require the client application (handled by the user) to execute orders. This interface is responsible for interacting with humans for execution of manual activities required for the workflow.

Interface 3 would deal with invoking outside applications; with respect to orders it would mean invoking processes in ancillary systems like Radiology, Laboratory etc. Interface 3 would deal with issues of notifying the other systems of orders (tasks) ready to be performed, scheduling the orders, checking the status of the tasks, and updating or modifying the status (due to ancillary systems/ applications). For example, when a Radiology order is executed, the RIS modifies the status of the order to 'done' and this change is reflected through the service interface.

Interface 5 would probably deal with the user (Placer), that would require the client application (handled by the user) to place orders.

Workflow Management Systems

A Workflow Management System provides procedural automation of a business process by management of the sequence of work activities and the invocation of appropriate human and /or IT resources associated with the various activity steps.

Healthcare involves several different business processes that are important to achieving its final goal i.e. providing care to the patient. But by far, the most important business process is the clinical workflow or the actual provision of care. When we talk about "task