IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4509

Design Patterns in Architectures Based on Use Cases

Alfredo Matteo and Christiane Metzner
Laboratorio Teoría y Tecnologías Orientadas a Objetos, Lenguajes y Sistemas TOOLS, Escuela de Computación, Facultad de Ciencias, Universidad Central de Venezuela, Apdo. 47764, Caracas 1041-A, Venezuela, amatteo/cmetzner@isys.ciens.ucv.ve

ABSTRACT

Object-orientation claims to promote reuse and therefore reduce development time and improve software quality. The reuse originally was considered at class level, that is, libraries of reusable classes. Progress in object technology gives today more importance to software architectures and design patterns. Any large object-oriented system should be built with design patterns. In this work we present an approach for developing an architecture based on use cases and the identification of some central design patterns. A case study exemplifying the proposed architecture is discussed.

1. INTRODUCTION

Progress in object technology and software engineering has made it possible to systematically create reusable application area architectures. Software architecture is emerging as a natural evolution of design abstractions involving the description of structural elements from which systems are built, interactions among those elements, patterns that guide their composition and constraints on these patterns. Historically, reusable software components have been procedure and function libraries (the 1960s) and in the late 60's, Simula67 introduced objects, classes and inheritance that resulted in class libraries. Both, the procedural and class libraries, are focused on code reuse. Since design is the main intellectual content of software and it is more difficult to create and re-create then code, design reuse should be more beneficial in economic terms. This has resulted in object-oriented frameworks and design patterns [2]. Besides specifying the structure and topology of the system, the architecture shows the correspondence between the system requirements and elements of the constructed system, thereby providing some rationale for the design decisions [8].

The OOSE method [4] and particularly use cases are a powerful approach for gathering early high level requirements: identifying user requirements, activities to be performed and user-system interactions. Under this method five different models are built: requirements, analysis, design, implementation and testing. The requirement and analysis models provide a description of the visible behavior of a system and could serve as a good basis for an object-oriented architecture.

In this work an architecture derived from the OOSE analysis model is proposed and expanded with Design Patterns as defined by [3]. This work is structured in the following sections: besides this introduction and the conclusions, in section 2 we describe briefly the essentials of frameworks and design patterns, in section 3 the proposed architectures based on use cases are discussed and a possible design with design patterns is presented and in section 4 an example using the architecture based on use cases is presented.

2. ESSENTIALS OF FRAMEWORKS AND DESIGN PATTERNS

An object-oriented framework is a reusable design for an application represented by a set of abstract classes and their collaborations. Together with the framework there is code that provides a default implementation. A framework then implies reuse of both code and design. It emphasizes those parts that will remain stable and the relationships and interactions among them and point out those parts that are likely to be customized for the application under development. In [3] or GoF Catalog, the concept of object-oriented design patterns was introduced, which can be used for describing parts of a design. A framework can contain a number of design patterns, but the opposite is never possible [2]. A generally accepted definition is, *a design pattern is a solution of a recurring problem in a certain context.* The GoF Catalog [3] contains 23 catalog entries that are quite domain independent, well-documented designs. They are specified using the following items:

- Motivation: a concrete scenario illustrates why a pattern is useful
- Abstract description of participating components and their interaction
- Guidelines regarding its application
- Code examples
- Cross-references to other design patterns

Design Patterns are a way of encapsulating developer's experience in a form that is easily communicated to other programmers and they provide among others: encapsulation experience, a common vocabulary and documentation of software designs.

3. ARCHITECTURES BASED ON USE CASES.

Software architecture encompasses the set of significant decisions about the organization of a software system [5], [1]: selection of the structural elements and their interfaces by which a system is composed, behavior as specified in collaborations among those elements, composition of these structural elements and behavioral elements into larger subsystem, architectural style that guides this organization

In the OOSE [4] method three kinds of objects are identified for each use case in the analysis model: interface, control and entity objects. As part of the analysis a robust architecture is defined which is resilient to change and fulfill the use cases. Some guidelines for control object identification are [4]:

- at least one control object can be assigned to each concrete or abstract use case
- each actor has at least an interface object through which he interacts with the system.

Following these guidelines a control object can be associated with each use case and an interface object can be associated to each actor that communicates with the system. The control objects serve as a "glue" between interface and entity objects. The entity objects may be persistent, in which case they have to be retrieved and stored in a data store.

From the above stated and using the schema for defining analysis objects in OOSE [9] an analysis model centered on the use cases identified for each human actor can be built: a central control object for each actor of the system, associated with control objects one for each use case the actor interacts with. A central interface object for each actor from where he can interact with the system and execute the use cases. Note that for non-human actors there will be no interface objects and entity objects only communicate with control objects.

The use of this schema for analysis model development is a way of handling complexity in large systems; partial analysis models are built one for each actor of the system. Clearly, entity objects that may intervene in more than one use case as well as control and interface objects may appear more than once, since more than one actor may interact with the same use case. Therefore it is necessary to factorize the partial analysis models when building the complete analysis model. Moreover, this factorization should be expressed by the software architecture definition as a traceability basis between analysis and design and is shown in Figure 1 where the control components are already depicted as classes.

In this architecture based on use cases the following software components can be identified: control, interface and entity components containing respectively the implementation of control, interface and entity objects, and a data store component encompassing the persistent objects of an application.

To obtain independence between the data store and the control com-

ponent a broker handling all services that require accessing the data store is introduced. The definition of this broker considers geographically non-distributed databases and / or files. For distributed applications, solutions with some leading technology as CORBA [10], DCOM [12] or RMI [11] should be considered.

For expressing the architecture the following considerations are made: suppose k human actors have been identified, j non-human actors and m use cases. There should be at least one human actor and one use case.

- The **entity component** encompasses all the implementations of the entity objects required for system execution.
- The m **control_usecase components**, one for each identified use case. They have at least the responsibility of requesting services from the broker and transmit the result of this service; they also communicate with the entity component were all active objects are located.
- The **broker** is a layer between the data and the application. He communicates only with control objects control_usecase, not necessarily with all of them.
- The k **control_humanactor** that receive the stimulus from an actor through the corresponding interface component and send a message to the appropriate control_usecase.
- The j **control_nonhuman** connecting directly to j components that represent non-human actors.
- The k **interface components**, one for each human actor.
- One **central control component** connected to k control_humanactor components.
- One **system interface component** that allows user identification and connects to the central control component.

The broker provides services to the clients independently of where the service is located and how it is implemented. The Facade pattern [3] intent is to provide a unified set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use. This intent is exactly what is needed in the broker's design; usually only one Facade object is required. Thus Facade objects are often Singletons [3]. A high-level diagram of the broker is shown in Figure 2. In the context of the architecture, the clients (control_usecase components) send requests to the broker (Facade pattern) which forwards them to the appropriate data store, that performs the actual work. The broker may have to do some internal work to translate its interface to the data store interfaces. For all the databases and / or files the same operations will be defined, as insert, delete, retrieve one or a set of elements but will be applied to different data stores. This can be adequately resolved by incorporating the Visitor pattern [3] which can be used when an object structure contains many classes of objects with different interfaces and the operations on these objects depend on their concrete classes. In Figure 3 the structure of the broker using the three design patterns Facade, Visitor and Singleton is shown.

Figure 1. An architecture based on use cases

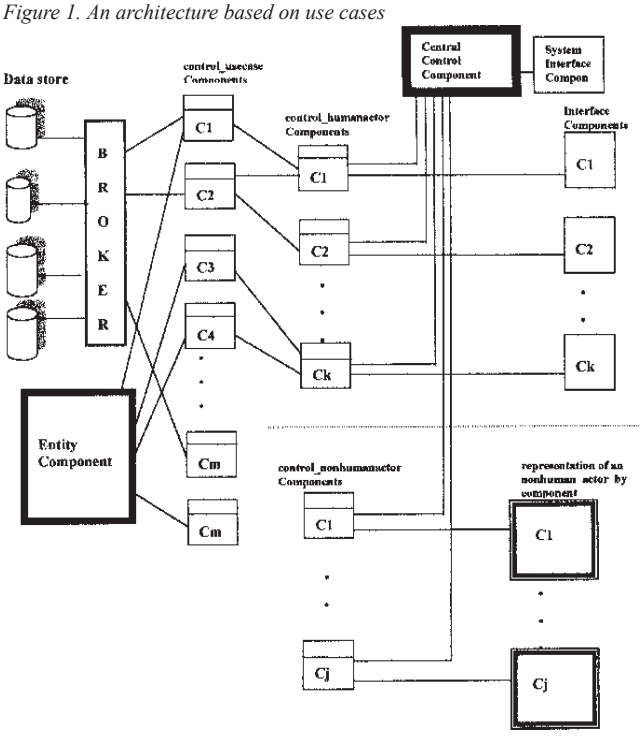
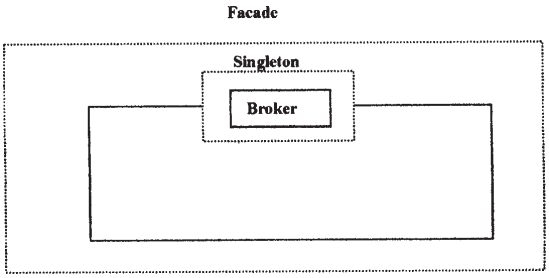


Figure 2. High-level design of the broker using design patterns



4. AN EXAMPLE

One of the goals of the Medical Science School at our University is to enable the diffusion of medical expert knowledge and use the technology to create a knowledge base for medical science education. The expert knowledge we are particularly interested in is focused on tropical diseases such as malaria or Chagas malady.

We started with three pilot applications to provide input for the case study [7]. The three applications are hypermedia tools that provide medical users with an environment allowing them to create their hypermedia documents. They are used in medical science education and the dissemination of medical expert knowledge in tropical diseases but they can be used in any other domain of medical sciences.

The tools developed also known as Software CAIBCO (Centro de Analisis de Imagenes Biomedicas Computarizadas, http://caibco.ucv.ve) are:

- Io: the digitizator and editor for creating the multimedia objects to be used in hypermedia documents.
- Zeus: the authoring tool used to create and annotate hypermedia documents
- Sphinx: the administration tool used to register information about the users (visiting users, registered users, Intranet users and system administrator) and objects in the system.

The development process used, defined in [5], is an iterative/incremental process that combines the methods OOSE [4] and OMT [6] maintaining the identity and flavor of each of the methods and taking advantages of the strong points of each one.

The authoring tool Zeus allows medical experts to create, annotate or update hypermedia documents using a web-based user interface (WUI). A hypermedia document consists of a set of pages. A page can be filled with multimedia objects (MMO) and subsequently connected to other pages. A

Figure 3. The Broker with design patterns

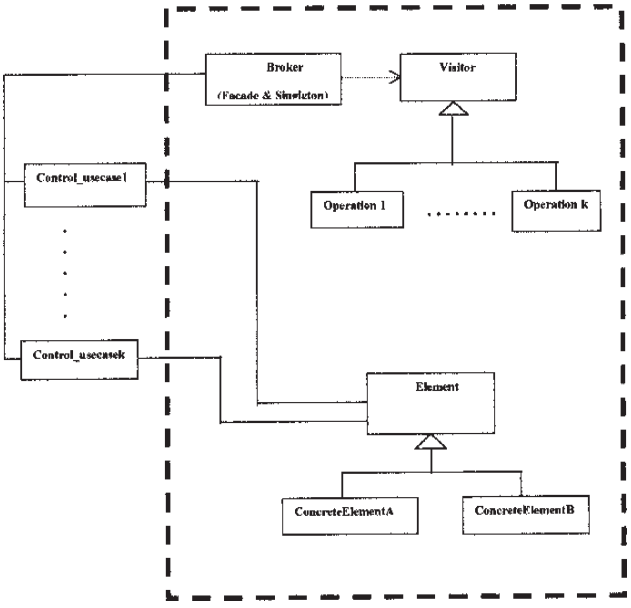
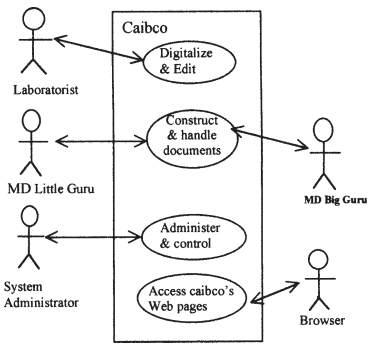


Figure 4. Use Case Model for CAIBCO



multimedia object is any piece of information such as audio, video, images or text and they have different capabilities and constraints.

Caibco‘s software has the following kinds of users: registered, visiting and Intranet / Internet users. Registered users are the following actors: *Big Guru MD, Little Guru MD, Laboratorist* and *Administrator*. Visiting users are those actors that can browse through hypermedia documents but are not allowed to create or update them and interment users are those that access the hypermedia documents using a standard market browser. The software has a WUI, a digitization GUI (Graphical User Interface) and an authoring WUI. Through these WUI's and GUI the actors communicate with the system. The digitization tool is not available on the Web; it may only be used at our installations. The *Laboratorist* is the actor in charge of the digitization and edition of images that can be used with the authoring tool. He feeds the system

with the images and their description. He may use any of the installed software for doing his job: Gscan, Capture, Imgwork, Moviemaker, and Movieplayer. New software for scanning and image edition may also be added. The *Big Guru Medical Doctor (MD)* is the actor whose expert knowledge will be incorporated into the system. This actor has great expertise in a particular medical area and although he may directly use the system to incorporate his knowledge, generally the *Little Guru MD* interacts with the Big Guru MD and the system. Both can be considered a specialization of an actor *MD.* He is in charge of collecting the expert knowledge, coordinates and decides which information and in what order will be put into the hypermedia documents The *Browser* is a non-human actor that interacts with the Caibco software and with a *Web Navigator*, a secondary actor that browses through existing and public hypermedia documents, typically students or physicians who want to learn or know something about a particular medical area. The *Administrator* is the actor responsible of all the administrative tasks related to registering and controlling the resources of the system. The use case model of Caibco‘s software is presented in Figure 4.

The MMO´s that can be used in the pages of a document, like X-rays, electrocardiograms (ECG), computerized tomography (CT) among others must have been digitized using the Io tool and stored in the MMO database. Information about these can be obtained through Sphinx.

Instantiating the broker with design patterns to our example the following diagram can be obtained:

5. CONCLUSIONS

Figure 5. Instantiation of the architecture for CAIBCO

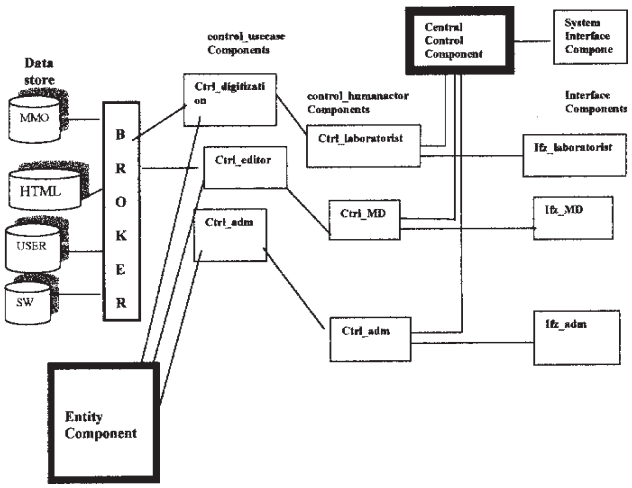
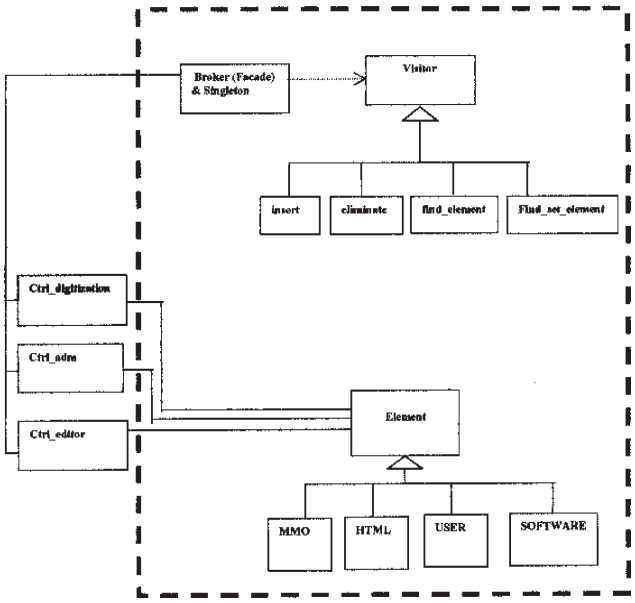


Figure 6. The Broker in CAIBCO's software



The use of the proposed architecture based on use cases and design patterns allows a team to focus on the problem rather then on implementation details, and through them efficient application of the object-oriented paradigm is understood and applied. From an architectural point of view, it is very important to find out how design objects are related.

If, as in our case, we had to start learning design patterns before applying them, an initial investment of time has to be considered before seeing results. We still have much more to learn about architectures, design patterns and frameworks before we can claim proficiency in capturing patterns from experts and sharing them, but it seems to be a way leading to productivity benefits worthwhile to be explored.

ACKNOWLEDGEMENT

This work has been partially supported by projects No. PG 03-13-4241-1998 from the CDCH and S1-95000512 Conicit.

REFERENCES

[1] Booch, G.; Kruchten, P.; *Software Architecture and the UML*, Tutorial 53W, OOPSLA´98, Vancouver, BC, Canada

[2] Mattson, M.; *Object-Oriented Frameworks,* University College of Karlskrona, / Ronneby, Sweden, 1996

[3] Gamma E.; Helm R.; Johnson R.; Vlissides J.; *Design Patterns. Elements of Reusable Object-Oriented Software*, Addison Wesley Publishing Co., 1994.

[4] Jacobson, I.; Christenson M.; Overgaard G.;*Object-Oriented Software Engineering, a Use Case Driven Approach*, Addison Wesley Publishing Co., 1992

[5] Aluen, M.; Arrechedera H.; Matteo, A.; Metzner, Ch.; *Developing a Web-based Object-Oriented Multimedia Medical System***,** The Thirty Second Hawai‘i Conference on System Science, HICCS-32, Maui, Hawaii, USA, January 1999

[6] Rumbaugh J.; Blaha, M.; Premerlani, W.;Eddy, F.; Lorensen W.; *Object-Oriented Modeling and Design* , Prentice Hall International Inc, 1991.

[7] Romero M.; *Procesos y Estrategias para incorporar Patrones de Diseño en Aplicaciones: Caso Estudio Software Caibco,* Tesis de Licenciatura, Universidad Central de Venezuela, 1998

[8] Shaw, M.; Garlan, D.; *Software Architecture Perspective on an Emerging Discipline*, Prentice Hall, New Jersey, 1996

[9] Losavio, F.; Matteo, A.; *Use case and multiagent models for object-oriented design of user interfaces*, Journal of Object-Oriented Programming, May 1997, Vol.10, No. 2

[10]Orfalli, Robert; Harkey, Dan *Client/Server Programming with JAVA and CORBA*.John Wiley & Sons. 1997.

[11]*Java Remote Method Invocation Specification*. Revision 1.4. JDK 1.1 FCS, February 10, 1997

[12]*Distributed Component Object Model Protocol. DCOM/1.0.* Network Working Group. November, 1996.