Proceedings of ICAD2006
4th International Conference on Axiomatic Design
Firenze – June 13-16, 2006
ICAD-2006-29

A USE CASE BASED OBJECT-ORIENTED SOFTWARE DESIGN APPROACH USING THE AXIOMATIC DESIGN THEORY

Andrey Ricardo Pimentel
andreyrp@cpgei.cefetpr.br
The Federal Technological University of Paraná – UTFPR
Graduate School in Electrical Engineering and Industrial Computer Science – CPGEI
Av. Sete de Setembro, 3165 - CEP 80230-901
Curitiba - Paraná - Brazil
Phone: +55 41 3310-4702 , Fax: +55 41 3310-4683

Paulo Cézar Stadzisz
stadzisz@lit.citec.cefetpr.br
The Federal Technological University of Paraná – UTFPR
Graduate School in Electrical Engineering and Industrial Computer Science - CPGEI
Av. Sete de Setembro, 3165 - CEP 80230-901
Curitiba - Paraná - Brazil
Phone: +55 41 3310-4759 , Fax: +55 41 3310-4683

ABSTRACT

The design of software systems has become a well-established discipline. Current software design methodologies and techniques enhance the software development processes contributing to improve software quality. However, creating a good software design solution still depends greatly on developers' expertise. It is difficult to evaluate if a software design solution is good enough without a theoretical foundation. The domain-free nature of the Axiomatic Design Theory, its high-level design concepts, and its theoretical foundation make the Axiomatic Design a powerful tool to be applied together with object-oriented software development methodologies and techniques, in order to help in guarantying the quality of the design solution.

This work presents an approach for object-oriented software design that intends to ensure the quality of the design solution along the development process. The goal of this approach is to propose and integrate methods that allow the use of the Axiomatic Design together with the Unified Modeling Language and the Unified Software Development Process.

Keywords: Axiomatic Design, Object-Oriented Software design, Unified Modeling Language, Unified Software Development Process

1 INTRODUCTION

As software became more and more important, new methodologies and techniques for software development have been continuously proposed and improved. The goal is making software development a more efficient and reliable process. Each new methodology or technique is an effort to make software development easier for developers, in terms of dealing with software complexity. The great challenge is to make bigger, better and easier to maintain systems, within the schedule and, more important, within limited budgets. To deal with these constraints, software development is becoming more an engineering process than an expertise based process.

Current software design notations, methodologies and techniques such as the Unified Modeling Language (UML) [1] and the Unified Software Development Process [2] are remarkable improvements to software development. They provide a unified notation for software design, better project organization using well-defined stages, artifacts and activities, and therefore, a better complexity management of the project.

However, according to Pressman, "even today, most software design methodologies lack the depth, flexibility and quantitative nature that are normally associated with more classical engineering design disciplines" [3]. Contributions are needed to help the designer to make design decision along the software development process. There are many design decisions that depends on the developers experience, especially choosing among alternative design solutions. The absence of a precise decision making criteria could easily make the process less reliable. Better design decisions are essential to make a better design.

The quality of the software product is easier to verify than is to achieve. According to Suh, "hardware, software, and systems must be designed right to be controllable, reliable, manufacturable, productive, and otherwise achieve their goals" [4]. A scientific basis for design will help to improve design activities by providing the designer with a theoretical foundation based on principles that characterize a good design solutions. This work presents an approach for software design that aims to guarantee the quality of the software design itself, based on the Axiomatic Design Theory [5]. The axiomatic design provides a theoretical foundation, based on axioms, theorems and corollaries, for helping the designer on deciding, which among many alternative design solutions is the "best" choice.

2 AXIOMATIC DESIGN CONCEPTS AND OBJECT-ORIENTED DESIGN CONCEPTS

Working in the mechanical engineering domain, at Massachusetts Institute of Technology (MIT), N. P. Suh created the Axiomatic Design Theory, in order to establish a scientific basis, with a theoretical foundation, for design activities [5]. One of its goals is to guarantee the quality of the design during the design process, enhancing the product's quality. It provides principles and laws to guide decision-making in design. The theory was established finding out common elements and attributes, which characterizes a good design. With those elements, the axiomatic theory provides the ability to judge the quality of design. This judgment ability helps the design process.

Copyright © 2006 by ICAD2006

According to Suh [4], the design process involves four different domains, as shown in Figure 1: the customer domain, the functional domain, the physical domain and the process domain [4].

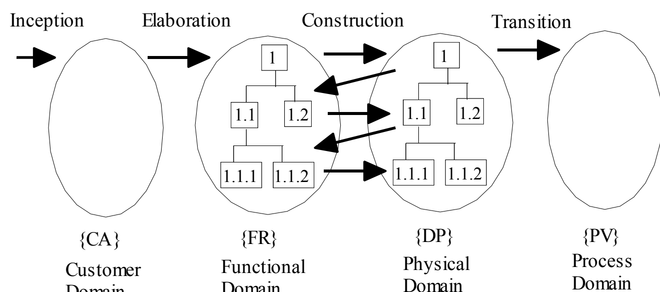

Figure 1 - The design domains and the Unified Process phases

The customer domain represents the customer needs by means of customer attributes (CA). The actual design activity is performed by mapping between the functional requirements (FRs) at the functional domain and the design parameters (DPs) at the physical domain. The process domain represents the means for realizing the product as process variables (PV).

The Unified Process cycle is composed of four phases: inception, elaboration, construction, and transition [2]. There is a close relation between the Unified Process phases and the Axiomatic Design domains.

In the inception phase a business model that describes the context of the system is made [2]. The business model created in this phase represents the customer needs for the system. From the point of view of the Axiomatic Design the customer needs described by the business model is inserted into the customer domain. Therefore, the business model resulting from the inception phase is represented in the customer domain.

One of the main goals of the elaboration phase is to capture most of the functional requirements (FRs) from the business model and formulating them as use cases [2]. At the functional domain of the Axiomatic Design the customer attributes are specified in terms of functional requirements (FRs) [4]. Therefore use cases perform the same role as the functional requirements (FRs) of the Axiomatic Design. This fact means that the elaboration phase creates a use case model, which represents the functional requirements (FRs) of the functional domain.

According to Jacobson, Booch and Rumbaugh [2] the construction phase is use case driven. In the construction phase, the system's classes, objects and their interaction are designed, modeled, implemented, and tested in order to realize the system use cases [2]. In the physical domain of the Axiomatic Design the design parameters (DPs) are created to satisfy the functional requirements (FRs) [4]. The system's classes, objects and their interaction can be considered design parameters (DPs) and are represented in the physical domain of Axiomatic Design.

In the transition phase, the product release is completed [3]. Among the main artifacts of this phase there are: the executable code itself, installations software, and architecture description. The process domain represents the process variables, which are the means to deliver the artifacts of the transition phase.

2.1 FUNCTIONAL REQUIREMENTS, DESIGN PARAMETERS AND CONSTRAINTS

According to Suh [4], "Functional Requirements (FRs) are a minimum set of independent requirements that characterizes the functional needs of the product". According to Jacobson, Booch and Rumbaugh [2] use cases "offer a systematic and intuitive means of capturing functional requirements (FRs) with a focus on value added to the user". A more detailed definition for use cases is presented in [1]. Use cases represents, at the Unified Process, the functional requirement (FR) concept from axiomatic design.

In a previous work Suh and Do in [6] and [4] an object-oriented software process was used and described. In this process, called Axiomatic Design for Object-Oriented Software System (ADo-oSS), a close relation between methods and attributes (component of object) and design parameters (DPs) was established. A use case based software lifecycle management model that uses axiomatic approach was presented in [7]. In this work was established a correspondence between use cases and functional requirements (FRs).

Design parameters (DPs) are the key variables that characterize the design that satisfies the specified functional requirements (FRs) [4]. Collaboration is a UML concept that represents the interaction between objects in order to realize a use case [1]. Collaborations can be used to specify the realization of a use case [1]. This means that collaborations can be used as design parameters (DPs) at higher abstraction levels of the design.

According to Booch, Rumbaugh and Jacobson [1], "classes are the most important building block of any object-oriented software system". A class describes a set of objects that share de same attributes, operations, and relationships. The classes and their instances (objects) that participate in collaborations to realize the systems use cases are design parameters (DPs). Class diagrams represent the structural model of the system classes. Interaction diagrams and state diagrams, among others, compose the dynamic model of the system classes [1].

Constraints are bounds on acceptable solutions. System constraints are constraints imposed by the system in which the design solution must function [5]. Functional requirements (FRs) are defined in order to deal with constraints. Similar to constraints, nonfunctional requirements usually need some kind of system functionality to be satisfied. These particular functionalities or infrastructure mechanisms can be modeled by use cases called "infrastructure use cases" [8].

2.2 DESIGN AXIOMS AND THE DESIGN MATRIX

The Axiomatic Design Theory is based on 2 axioms, related theorems and corollaries [5]. The Axiomatic theory states that if the design satisfies these axioms, it can be considered a good design. The design axioms are stated as follows:

Axiom 1: The Independence Axiom. "Maintain the independence of the functional requirements (FRs)" [5].

Axiom 2: The Information Axiom. "Minimize the information content" [5].

The Independence Axiom states that the functional requirements (FRs) must always be maintained independent of one another by choosing appropriate design parameters (DPs).

Copyright © 2006 by ICAD2006

During the design activity, the design parameters (DPs) are conceived from the functional requirements (FRs). Each functional requirement (FR) is associated with those design parameters (DPs) that satisfies it. Two functional requirements (FRs) are dependant if a design parameter (DP) that is used to satisfy a functional requirement (FR), is also used to satisfy the other functional requirement (FR).

In large information systems, several development teams frequently split the development task. Frequently, these teams are spread out across the world. In this context, it is very important to reduce the dependence between the teams to minimum. The Independence Axiom applied to the system functional requirements (FRs) will helps to guarantee that a change in a design parameter (DP) that satisfies a functional requirement (FR) will not affect others functional requirements (FRs). For these reason, if the design satisfies the Axiom 1 it will be easier to coordinate the efforts of various development teams.

The association between each functional requirement (FR) and its correspondent design parameters (DPs) is mapped in a matrix called design matrix. The functional requirements (FRs) are represented as design matrix lines and the design parameters (DPs) as its columns. The relation between a functional requirement (FR) and a design parameter (DP) is represented as a non-zero element of the matrix, as shown in Figure 2. A design matrix element equals to zero means that the corresponding design parameter (DP) and functional requirement (FR) are not related.

From the point of view of Object-Oriented software design, the design matrix represents the relation between the use cases and the collaboration of objects that realize them.

$$\begin{bmatrix} A_{11} & 0 & 0 \\ 0 & A_{22} & 0 \\ 0 & 0 & A_{33} \end{bmatrix} \qquad \begin{bmatrix} A_{11} & 0 & 0 \\ A_{21} & A_{22} & 0 \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \qquad \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

Uncoupled Design   Decoupled Design   Coupled Design

Figure 2 - Uncoupled, Decoupled and coupled designs

The independence of the functional requirements (FRs) can be viewed and measured with the design matrix. According to Suh in [5], there are three main types of design with respect to the independence of the functional requirements (FRs), as shown in the design matrices of Figure 2: uncoupled, decoupled, and coupled design [5].

The uncoupled design means that each functional requirement (FR) is satisfied by only one design parameter (DP). This type of design satisfies the Independence Axiom and can be considered a good design. The decoupled design matrix represents an acceptable design solution, where there is an order for satisfying the functional requirements (FRs). The coupled design matrix represents a bad and unacceptable design, which doesn't satisfy the Independence Axiom. It is a matrix that cannot be turned into a decoupled one by changing its lines.

Only the use of Object-Oriented techniques such as use cases, classes and objects does not guarantee a good design. In the example shown in figure 3, for a hypothetical sales system, the use cases "Register Customer", "Register product", and "Register invoice" are found. In order to realize these use cases are created the following classes: an user interface class, called "RegisterForm", that is used for all registrations, a control class, called "ControlRegister" that controls all registration processes, and an entity class, called "Register" that stores in memory and handle all the information for the system. This design is clearly a coupled design.

FRs \ DPs	RegisterForm class	ControlRegister class	Register class
Register customer	X	X	X
Register product	X	X	X
Register invoice	X	X	X

Figure 3. A Coupled design for an object-oriented software

According to Jacobson, Booch and Rumbaugh [2] prioritize use cases is one of the activities of the Unified Process. The goal of this activity is to determine which use cases can be realized (analyzed, designed, implemented and tested) in early interactions and which can be realized later [2]. If an object-oriented software design solution is a decoupled one, according to Suh, When a design is decoupled "the independence of the functional requirements (FRs) can be guaranteed if and only if the design parameters (DPs) are determined in a proper sequence" [4]. Therefore, design matrix can guide the prioritization of the use cases in order to keep the independence of the use cases.

3 FUNCTIONAL DECOMPOSITION FOR OBJECT-ORIENTED SOFTWARE DESIGN

According to Jacobson, Booch and Rumbaugh, the Unified Process is use case driven. This means, "the development process follows a flow that derives from the use cases" [2]. Use cases are identified and specified in terms of its flows of events. These flows of events can be specified in terms of a sequence of interactions between the system and the actors. These interactions can be specified as a sequence of interaction between objects using interactions diagrams. This refinement mechanism is very close to the decomposition of the functional requirements (FRs), used by Axiomatic Design. Based on this, a hierarchy for functional decomposition can be established.

In the Axiomatic approach, the functional requirements (FRs), not the system modules, are decomposed in a hierarchical structure. Decomposing functional requirements (FRs) instead of decomposing modules of the system enhances the analysis activity allowing the identification of a greater level of details of the system in early stages. This also permits validating the identified functional requirements (FRs) in terms of the two axioms to verify the quality of the design early in the design process.

3.1 DECOMPOSITION LEVELS

In the proposed design approach the decomposition of functional requirements (FRs) will be divided into four different levels. At the first level, the use cases are defined. The flows of

Copyright © 2006 by ICAD2006

events are obtained at the second level. The activities are the result of the third level of the decomposition and in the fourth level technical services are obtained. The number of decompositions at each decomposition level is not defined, but it must be enough to satisfy the design needs for that abstraction level. A model of the proposed functional hierarchy is shown on Figure 4.

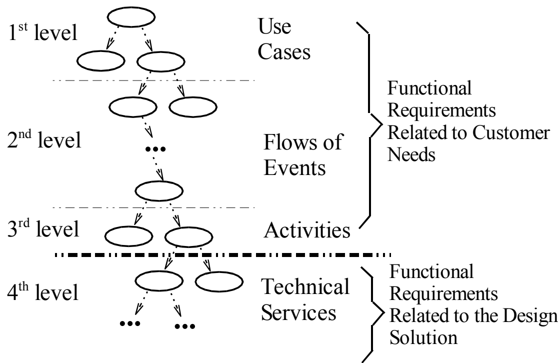

Figure 4 - Levels of the Proposed Functional Hierarchy

The functional requirements (FRs) will be represented by different UML concepts, according to their level at the functional hierarchy. The functional requirements (FRs) will, also, be classified according to the type of the required service. According to the hierarchy level, functional requirements (FRs) will be represented by: use cases, flows of events, activities, and technical services. A use case can be defined as a complete utilization of the system. A flow of events can be defined as a sequence of interactions between the system and an actor [2] and is a part of a use case functionality. An activity is one event of the flow of events. A technical service can be defined as a service (i.e., a piece of a functional requirement (FR)) expected from an object to another. The technical service concept is very close to responsibility of a class defined by Beck and Cunningham in [9] where responsibility identifies a problem to be solved by an object.

During the design process, a design parameter (DP) is created to satisfy a given functional requirement (FR). At the first level of decomposition the functional requirements (FRs) are represented by use cases. In this level, the design parameters (DPs) are represented by collaborations. At the second and third levels, when the functional requirements (FRs) are represented by flows of events and activities, the design parameters (DPs) are also represented by collaborations. In the fourth level, where a functional requirement (FR) is represented by a technical service, the corresponding design parameter (DP) is represented by an object of a class.

3.2 THE DECOMPOSITION PROCESS

The design activities for each level of decomposition are similar. The functional requirements (FRs) are identified, and then the design parameters (DPs) are created to satisfy those functional requirements (FRs). The relations between them are represented in the design matrix. Then, the design is evaluated in terms of Axiom 1 and in terms of Axiom 2. If the design is good then a new decomposition is started. If the design has reached all information details needed by the current level then it goes the next decomposition level. If the obtained design solution is not good according to the two axioms then a new design solution should be found either by creating a new set of design parameters (DPs) or choosing a new set of functional requirements (FRs).

From the point of view of the Axiom 1, a design solution is considered a bad or unacceptable solution when it is a coupled one. In this case a new design solution should be created. The best design solution is an uncoupled design. A decoupled design can be considered a satisfactory one. However, the designer could want to find a better design and for this, establish an alternative solution. An alternative statement for the Independence Axiom is "Of two feasible designs, the one with higher functional independence is superior" [5]. Modifying the current design in a few points can create an alternative solution. The evaluation of which design solution is better can be done with the application of Axiom 1 calculating the reangularity (see Equation 1) of the possible solutions.

$$R=\prod_{\substack{i=1,n-1\\ j=1+i,n}}\left(1-\frac{\left(\sum_{k=1}^{n} A_{ki}A_{kj}\right)^{2}}{\left(\sum_{k=1}^{n} A_{ki}^{2}\right)\left(\sum_{k=1}^{n} A_{kj}^{2}\right)}\right)^{1/2} \quad (1)$$

The independence between functional requirements (FRs) can be evaluated by a quantitative measure, called reangularity [5]. Reangularity measures the orthogonality between the design parameters (DPs) and is calculated as shown in Equation 1 defined by Suh in [5]. The meaning of the reangularity metric is better explained in [5]. This measure has the value 1 when the design matrix is uncoupled, and 0 when coupled.

For Reangularity calculation purposes, an "X" in a cell of the design matrix corresponds to a numeric value equals to 1. In this way, an empty cell corresponds to a 0.

The Unified Process iterative and incremental and its construction phase is said to be "use case driven". The construction phase is divided in iteration. The use cases of the system are prioritized and ordered to establish which group of use cases can be realized at each iteration of the construction phase. In every iteration, a group of use cases that fulfill a system's functionality is analyzed, designed, implemented and tested and then start another iteration with other group of use cases [2].

The decomposition of the functional requirements (FRs) in software design follows the basic principles of the Axiomatic Design but has to be adapted to the needs of software design. The decomposition process of the Axiomatic Design, illustrated in Figure 5, creates the hierarchy fulfilling each level before passing to the next one. A use case driven interactive and incremental process could also, needs to decompose a branch of hierarchy tree totally in depth before going to others branches. This depth-first approach is possible if the designer maintains each decomposition level design matrix coherent to the previous level [10]. This means that if a relation between a functional requirement (FR) and a design parameter (DP) appears and this relation is not coherent with the previous levels, the designer must review the design at those levels [10].

Copyright © 2006 by ICAD2006

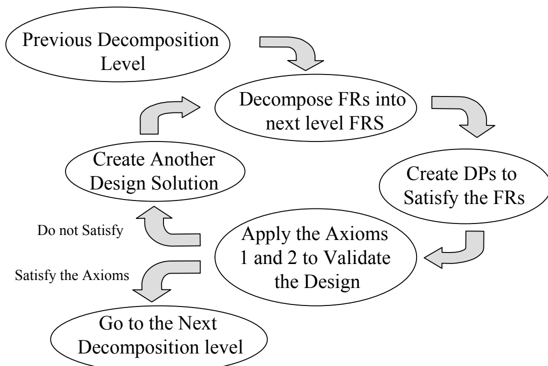

Figure 5 - Proposed Process Activity Flow

3.3 FIRST LEVEL OF DECOMPOSITION

At the first level of decomposition, functional requirements (FRs) represented by use cases will be mapped into design parameters (DPs) represented by collaborations and this will be represented in the design matrix. Each non-zero element of the design matrix will represent that a collaboration is related to the use case. It will be a full relation when the whole collaboration is associated to the use case or it will be a partial relation when only part (roles) of the collaboration is related.

The design matrix represents the use cases in the lines and the respective collaborations in the columns. The matrix cells represent the relations between the use cases and the collaborations. An "X" at the cell relating a given use case and a collaboration, indicates that this collaboration realizes the use case. It is also possible to represent partial dependencies in the design matrix. The "X" at the cell relating a given use case and a collaboration that realizes another use case, indicates that the realization of the use case requires a method of a class used in this collaboration. This indicates that these two use cases are dependent one of another.

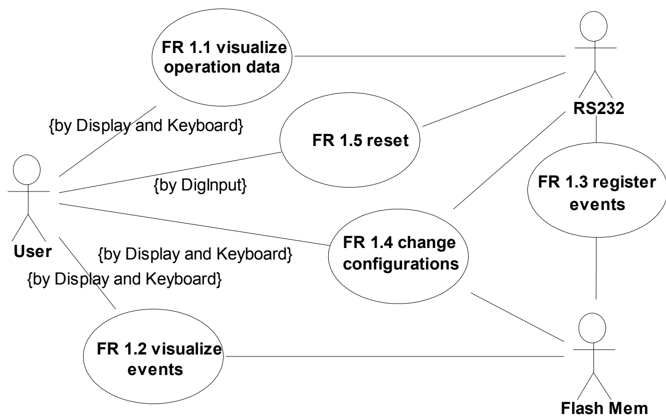

Figure 6 - Use cases for the case study system

As an example, the design of an embedded system for eSysTech eAT55's evaluation board[1] hardware is analyzed. The use cases for this example are identified and shown in Figure 6. The design matrix, shown in Figure 7 represents the relations between the use cases and the collaborations for the system. The "X" at the cell relating the use case "Register Events" and "Collaboration Visualize Operation Data", indicates that the realization of the use case requires a method of a class used in this collaboration. This means that the use case "Register Events" depends on the realization of the use case "Visualize Operation Data". "Visualize Operation Data" realization is also used for the use case "Register Events" realization. This fact means that "Visualize Operation Data" has to be realized before "Register Events".

FR \ DP	DP 1.1 Collaboration Visualize Operation data	DP 1.2 Collaboration Visualize Events	DP 1.3 Collaboration Register Events	DP 1.4 Collaboration Change Configuration	DP 1.5 Collaboration Reset
FR 1.1 Visualize Operation data	X				
FR 1.2 Visualize Events	X	X			
FR 1.3 Register Events	X	X	X		
FR 1.4 Change Configuration	X	X	X	X	
FR 1.5 Reset				X	X

Figure 7 - First level Design Matrix for the system

The dependence between the use case "Register Events" and the use case "Visualize Operation Data" can be minimized if an infrastructure use case is identified and factored from the two other use cases. Infrastructure use cases are not instantiated but included or extended by other use cases. In this case it is possible to identify the infrastructure use case "read serial port". The design matrix for this alternative solution is shown in Figure 8.

FR \ DP	DP 1.6 Collaboration Read Serial Port	DP 1.1 Collaboration Visualize Operation data	DP 1.4 Collaboration Change Configuration	DP 1.2 Collaboration Visualize Events	DP 1.3 Collaboration Register Events	DP 1.5 Collaboration Reset
FR 1.6 Read Serial Port	X					
FR 1.1 Visualize Operation data	X	X				
FR 1.4 Change Configuration		X	X			
FR 1.2 Visualize Events		X	X	X		
FR 1.3 Register Events	X		X		X	
FR 1.5 Reset			X			X

Figure 8 - First level Design Matrix for the alternative solution

The two design solutions have to satisfy the Axiom 1. Both of them are decoupled, which means they are satisfactory design solutions. In order to help to decide which one is better, it is possible to calculate the reangularity of the design matrices. The reangularity is calculated using Equation 1. The value for reangularity of the first design solution is 0,0114. The obtained value for the alternative design solution is 0,1018. A greater value

[1] "eSysTech eAT55 Evaluation Board" is a trade mark of eSysTech Embedded Systems Technologies. www.esystech.com.br

Copyright © 2006 by ICAD2006

for reangularity means a less coupled and better design, according to Axiom 1. In terms of the Independence Axiom, the alternative solution is possibly better than the first one. These two solutions must also be evaluated in terms of the Information Axiom.

3.4 SECOND AND THIRD LEVELS OF DECOMPOSITION

At the second decomposition level, the functional requirements (FRs) are represented by flow of events, elicited from the use case specification. The flows of events describe a use case. The flows of events is obtained through the description of the interaction between an actor and the system as described in [2], [8] and [14].

Once the functional requirements (FRs) were found, its necessary to create the design parameters (DPs) to satisfy them. At this decomposition level, the design parameters (DPs) will be represented by collaborations between objects. For each flow of events, one collaboration was created. The collaborations at this level will represent a part of the corresponding collaboration of the previous level.

At the third level of decomposition, the flows of events were decomposed into activities. Each activity represents one interaction between the actor and the system. As a design parameter (DP), one collaboration between objects was created for each activity (functional requirement (FR)).

3.5 FOURTH LEVEL OF DECOMPOSITION

The activities were decomposed into technical services at the fourth level of decomposition. The design parameters (DPs) for this level of decomposition are represented by objects (i.e. instances of software classes). For each technical service, one object was created and this relation was mapped into the design matrix. The relations between each technical service and the others objects that were used to satisfy it were mapped into the matrix. Therefore each line of the matrix represents a technical service and the objects that take part on the collaboration that satisfies. The definition and the representation of the interactions between objects is an important artifact in an Object-Oriented software design. The representation of these interactions by the UML is done using the sequence diagrams or the communication diagrams [1].

At the fourth level of decomposition, the lines of the design matrix are closely related to the sequence diagram construction. Considering the design matrix shown in Figure 9, the technical services, "FR 1.6.1.1.1 control read port", "FR 1.6.1.1.2 read port", "FR 1.6.1.1.3 receive new operation data", and "FR 1.6.1.1.4 receive new event" are a result of the decomposition of the functional requirement (FR) (activity) "FR 1.6.1.1 read port". The objects (design parameters (DPs)) that satisfy those technical services will be those that appear on the sequence diagram for the activity "FR 1.6.1.1 read port" as shown in Figure 10.

FRs \ DPs	DP 1.6.1.1.1 object from class CCtrlReadSerialPort	DP 1.6.1.1.2 object from class CIntRS232	DP 1.6.1.1.3 object from class COpData	DP 1.6.1.1.4 object from class COpEvent
FR 1.6.1.1.1 control read port	X			
FR 1.6.1.1.2 read port	X	X		
FR 1.6.1.1.3 receive new operation data	X		X	
FR 1.6.1.1.4 receive new event	X			X

Figure 9 – Partial 4th level design matrix

The design matrix is a useful tool for identifying and creating the design parameters (DPs) and on guarantying the functional independence of the functional requirements (FRs). Beside this utility, the design matrix can help the designer at the task of creating the UML diagrams such as sequence diagrams, which are an important part of an Object-Oriented software design.

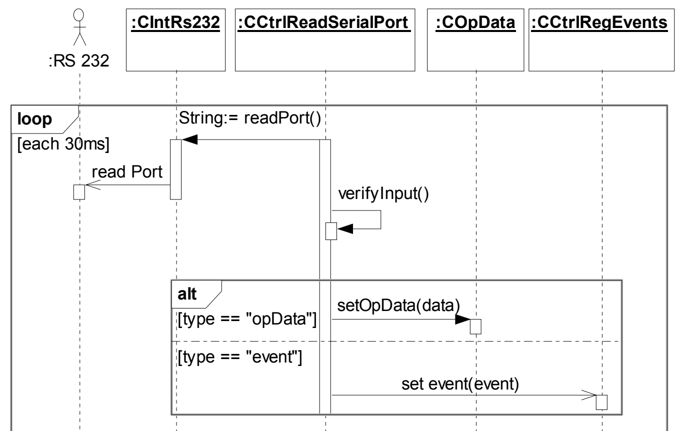

Figure 10 - Sequence diagram for "FR 1.6.1.1 read port"

4 INFORMATION CONTENT OF SOFTWARE DESIGN

In order to validate the design quality according to the information axiom, it is necessary to calculate the information content for the design. Suh suggests that the information content of software systems is related to the complexity of the software [4]. Information content is defined as the inverse of the probability that a design parameter (DP) would satisfy a related functional requirement (FR). The more complex a software system is, more difficult is to design and implement it with success.

To calculate the information content for a software system design it will be used metrics of software design complexity. The chosen metrics should be easy to calculate and well established in the software industry. Different software design concepts are used to represent functional requirements (FRs) and design parameters (DPs) at the different levels of the functional

Copyright © 2006 by ICAD2006

hierarchy. For this reason, different software complexity metrics will be used for each level of the hierarchy.

The most important design concept at the first level of decomposition is use case. There is a software complexity metric called use case points that is based on use case [11]. The use case point measures the functional complexity of the actors and the use cases of the system. The information content will be calculated by the ratio of the use case points counting (*ucpc*) for the system and the estimation for the use case points for the system based on historical data of the organization (*eucp*). The resulting equation is shown in Equation (2). This equation will be used to calculate the information content at the first and the second level of decomposition.

$$I = \log(\frac{ucpc}{eucp}) \quad (2)$$

The information content of the third level of decomposition can be calculated using function points [12]. In an equation, shown in Equation (3) and similar to the Equation (2), the information content will be calculated by the ratio of the function points counting (*fpc*) for the system and the estimation for the function points for the system based on historical data of the organization (*efp*).

$$I = \log(\frac{pfc}{epf}) \quad (3)$$

At the fourth level of decomposition, an object-oriented complexity metrics should be used. The suite of metrics proposed by Chindamber and Kemerer measures object-oriented software characteristics [13]. The metrics are: weighted methods per class (WMC), depth of the inheritance tree (DIT), number of children (NOC), coupling between objects (CBO), response for a class (RFC) and lack of cohesion of methods (LCOM) [13].

The information content for each metric is calculated with the ratio between the metric value for the class and the estimation for this value based on historical data of the organization. The resulting equations are shown in Equations (4), (5), (6), (7), (8) and (9).

$$I_{WMC} = \log(\frac{WMC}{EWMC}) \quad (4)$$

$$I_{DIT} = \log(\frac{DIT}{EDIT}) \quad (5)$$

$$I_{NOC} = \log(\frac{NOC}{ENOC}) \quad (6)$$

$$I_{CBO} = \log(\frac{CBO}{ECBO}) \quad (7)$$

$$I_{RFC} = \log(\frac{RFC}{ERFC}) \quad (8)$$

$$I_{LCOM} = \log(\frac{LCOM}{ELCOM}) \quad (9)$$

$$I_{classe} = \alpha I_{WMC} + \beta I_{DIT} + \delta I_{NOC} + \gamma I_{CBO} + \theta I_{RFC} + \omega I_{LCOM} \quad (10)$$

The total information content for a class is calculated by a weighted sum, as shown in Equation (10). The weights will be given by the designer experience.

As an example, the information content calculation for a class of a given working hours registering system will be shown. In Figure 11, two classes are presented: "CIntBD" and "CActivity". The information content for classes "CIntBD" and "CActivity" is calculated by the sum of the information content of each class. For each class, the information content is calculated by the sum of the information content of each metric. The obtained value for class "CIntBD" is $I_{CIntBD} = -0.530$ and for class "CActivity" is $I_{CActivity} = -0.585$. A negative value for the information content represents a decrease on the total information content. This fact can indicate that these classes could be a good solution. But, the solution can only be considered good when the information content for all classes of the system has been calculated.

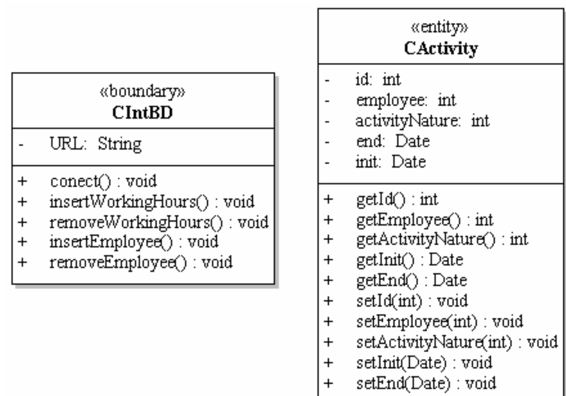

Figure 11 - "CIntBD" and "CActivity" Classes

Use case points and function points are well-established metrics of software functional complexity. The use case points and the function points for a system can be counted with the help of a CASE software tool. The Chindamber and Kemerer metrics suite can be easily calculated during the design activity using the class diagrams and the interaction diagrams and later by the analysis of the code of the classes.

5 CONCLUSIONS

This work presented an approach for object-oriented software design that intends to ensure the quality of the design solution along the development process. The goal of this approach is to propose and integrate methods that allow the use of the Axiomatic Design together with the Unified Modeling Language and the Unified Software Development Process.

This design approach has defined object-oriented software design concepts and their correspondence with Axiomatic Design concepts. Relationships between object-oriented software design concepts such as use cases, collaborations, classes, operations, and objects and functional requirements (FRs), design parameters (DPs), and the zigzagging process were defined. Relationships between the four domains of Axiomatic Design and the phases of the Unified Software Development Process were established in order to facilitate their integrated use.

A functional decomposition hierarchy, strongly based on the use case concept, has been defined. The proposed approach defines the main stages, activities, and artifacts and how they are applied. It describes the application of the Independence and the Information axioms. The proposed approach helps the design to maintain a quality of the software design due to the application of the Axioms 1 and 2.

Copyright © 2006 by ICAD2006

The proposed approach is closely related to the utilization of the system by an actor (human, software or hardware). For this reason it helps to maintain a correspondence between the functional requirements (FRs) from lowers levels and the systems use cases. This correspondence with the use cases helps the application of the use case driven, iterative and incremental life cycle of the Unified Software Development Process.

A framework for information content calculation, specific for object-oriented software design was described. This framework uses object-oriented software complexity metrics from literature, such as use case points and the CK metrics suite.

6 ACKNOWLEDGEMENTS

The present work was supported by the National Council for Scientific and Technological Development (CNPq) – Brazil.

7 REFERENCES

[1] Booch, G., Rumbaugh, J., Jacobson, I., 2005, The Unified Modeling Language User Guide, 2nd Edition, *Addison Wesley*

[2] Jacobson, I.; Booch, G.; Rumbaugh, J., 1998, The Unified Software Development Process, *Addison Wesley*

[3] Pressman, R. S., 2005, Software Engineering: A practitioner's approach, 6th ed, *McGraw-Hill*

[4] Suh, N. P., 2000, Axiomatic Design: Advances and Applications, *Oxford University Press*

[5] Suh, N. P., 1990, The Principles of Design, *Oxford University Press*

[6] Suh, N. P.; Do, S., 2000, Axiomatic Design of Software Systems. *In: CIRP Annals. Vol. 49, n. 100, p. 95-100*

[7] Do, S. H., 2004, Software Product Lifecycle Management Using Axiomatic Approach. *In: Proceedings of the 3rd International conference on axiomatic design – ICAD2004*

[8] Jacobson, I., Ng, P. W., 2004, Aspect-Oriented Software Development with Use Cases. *Addison Wesley*

[9] Beck, K., Cunningham, W, 1989, A Laboratory for Teaching Object-Oriented Thinking, *in: Proceedings of the OOPSLA'89*

[10] Tate, D., 1999, A Roadmap for decomposition: Activities, Theories, and Tools for System Design. *PhD thesis, Department of Mechanical Engineering at Massachusetts Institute of Technology*

[11] Anda, B.; Dreiem, D.; Sjøberg, D.; Jørgensen, M., 2001, Estimating Software Development Effort Based on Use Cases - Experiences from Industry. *In: UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools, 4th International Conference*

[12] Albrecht, A. J., 1979, Measuring Application Development Productivity. *In: Proc. IBM Application Development Symposium, pp 83-92*

[13] Chindamber, S. R.; Kemerer, C. F., 1994, A Metrics Suite for Object-Oriented Design. *In: IEEE Transactions on Software Engineering, vol. SE-20, no. 6*

[14] Bittner, K., Spence, I., 2003, Use CaseModdeling, *Addison Wesley*

Copyright © 2006 by ICAD2006