Using XML and Frameworks to develop Information Systems

Toacy C. de Oliveira,

Ivan Mathias Filho,

Carlos J.P. de Lucena

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rua Marquês de São Vicente 225

Rio de Janeiro, RJ, 22453-900, Brazil

Email: {toacy,ivan,lucena}@inf.puc-rio.br

PUC-RioInf.MCC24/01 July, 2001

Abstract

To accomplish the software development time and cost constraints this development should take place in an environment that helps the designer to deal with the large amount of concepts obtained during the domain analysis phase and the semantic gap between those concepts and the object oriented design model due to their different levels of abstraction. This paper describes the main features of an environment designed to support the development of IS software based on framework reuse and XML specifications.

Keywords: Reuse, Framework, Domain Analysis, XML, Object-Oriented.

Resumo

Para que o desenvolvimento de software atenda as restrições de prazo de conclusão e orçamento é necessário que o desenvolvimento ocorra em um ambiente que auxilie o projetista a lidar com uma grande quantidade de conceitos obtida durante a etapa de Análise de Domínio, além do *gap* semântico entre tais conceitos e o design orientado a objetos, que ocorre em razão dos diferentes níveis de abstração existentes. Este trabalho descreve as principais características de um ambiente projetado para dar suporte ao desenvolvimento de Sistemas de Informação baseado na reutilização de frameworks e em especificações XML.

Palavras-chave: Reutilização, Frameworks, Análise de Domínio, XML, Orientação a Objetos.

1 Introduction

The cost and time-to-market constraints imposed on modern software development oblige application designers to leave the *made from scratch* approach and adopt a reuse enabled support to software development. As a consequence during the system development proven solutions such as Components [6] and Frameworks [7] must be composed with an application initial specification to obtain the final design/code. It is also important that the act of achieving this application initial specification be handled by a process that captures domain knowledge and guides the application designer to map/trace its translation to any design representation, such as Object Oriented Design , from where the final specification can be extracted.

In this paper we report the ongoing development of an environment that uses a Domain and Reuse Driven approach to the software development problem. This work is an extension of the approach presented in [19] with the introduction of the XML/XMI standards [10] to represent the designs involved. Another change to the approach is the use of a framework design language to be able to deal with generic framework specifications.

It is also important to mention that such an approach should be based upon some characteristics:

⇒ Compatibility - It must use market standards to provide compatibility/integration with other systems.

⇒ Code Legibility - During development the compilation/debugging is usually done with a market IDE such as Borland JBuilder and IBM VisualAge, so the user must understand the final code.

⇒ Focused on OO - The user must only know OOP techniques

⇒ UML - Due to OMG standards.

⇒ Upgradable - Reuse actions such as inheritance, composition, patterns, frameworks and aspects can evolve.

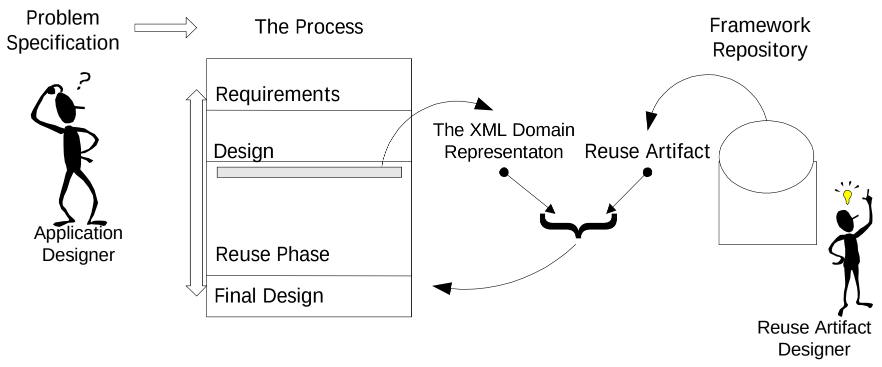

Figure 1 - The approach overview

With these characteristics in mind our approach adopts market standards like UML and XMI that are used as the basis of the representation of the diagrams involved. The approach begins by making a thorough analysis to determine the common and different aspects of the domain with the FODA Method [4] and Use Cases [13] to reduce the so-called “Semantic Gap.” After that the application designer creates a class diagram based on the previous models that will be used as the application initial specification.

In the reuse phase we have a modification of the traditional software development approach where reuse should be handled. From this phase a XMI representation of the domain specification is obtained to facilitate the composition manipulation of this specification with the reuse specification (also expressed in XMI) that is stored in a reuse repository (see Figure 1). After the choice of the reuse specification, which is done empirically, the environment user will be guided to execute reuse actions, stated as class redefinition clauses, pattern usage or composition that have been defined by the reusable artifact designer.

Section 2 describes the approach overview and its adaptations to the current processes. In Section 3 we describe how domain information is collected. In Section 4 the reuse approach is depicted. Section 5 reports on a Rental System development used as a case study. In the last section we present our conclusions and future directions.

2 The Approach Overview

The major software development approaches in use today [13][11][12] state that to obtain a good representation of an application, the designer should collect the behavior and structure of the application by using a repetitive and incremental approach. This approach begins with the Requirement Analysis Phase where the functional and non-functional requirements of the system are collected to represent its main functionality (from the end user's point of view). After that the application designer moves to the Design Phase to represent the functionality by using an object-oriented representation. In this phase the designer represents the main structures of the system and how they collaborate to achieve the systems functionality. In the last[1] phase, the Code Phase, the object-oriented structure is represented in some programming language. Usually, the designer moves back to the first phase to begin another development cycle and to reach the desired level of application representation.

Our approach introduces an explicit Reuse Phase in the classical process (see Figure 1). In this phase the application designer must identify the possibility of reuse of an object-oriented framework that represents a proven solution to the application domain he is working on. To achieve this reuse, the designer should search the Reuse Artifact Repository to find if a reuse artifact can be used. This phase occurs between the Design Phase and the Code Phase. Once the reuse artifact is chosen, the application designer will be guided to perform a set of reuse actions that were defined by the reuse artifact designer. It is important to say that the match between the designer's application specifications — that is, the application OO Design — and the reuse artifact stored in the repository is completely manual.

To represent the application design and the reuse design, we have chosen the Feature Oriented Domain Analysis Method, FODA, in combination with the Unified Modeling Language, UML, to achieve a more precise representation of the application semantics and provide compatibility with the market standards. Since our approach combines O.O. designs, we're particularly interested in the class diagram and how to obtain it. Naturally, class diagrams are a structural refinement of domain designs that in our approach are obtained combining Features Diagrams and Use Cases in a guided way, to provide traceability. Once these UML class diagrams contain a good representation of the domain functionality and structure we can use them as the source for the composition (reuse)

[1] Maintenance and test are not relevant to the scope of this paper.

step. The problem now is to obtain a representation of those diagrams that can be manipulated by programs. This can be achieved through the use of a XML representation called XMI.

XMI is an acronym for XML Metamodel Interchange, which is an attempt of the OMG/W3C organizations to provide a platform independent representation of UML designs. For that they propose a complete representation of UML using the XML format that can be shared among software vendors [10]. The XMI representation of a design can be achieved using case tools like Argo/UML [2] or using the IBM XMI toolkit converter for Rational Rose .

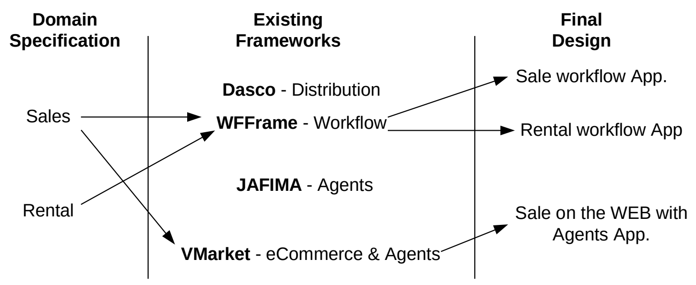

Figure 2 – Domain Analysis and Composition (reuse) representation.

An important point in our approach refers to the initial design boundary. To obtain a high level of reuse we propose that the designer initially should specify the core functionality and structure of his application using Domain Engineering techniques such as Domain Analysis and Domain Design [3]. With this approach we tend to isolate the application domain design from technological issues, such as WEB enabled, agent usage, distribution and so on, which tend to be the most variable part. After that the designer executes the composition step, that is the framework reuse. For example, in Figure 2 we have the specification of two domains, Sales and Rental, that can be composed with existing frameworks such as WFFrame [19], Dasco [8], VMarket and JAFIMA [6].

3 Obtaining Domain Information

3.1 Domain Analysis

The growing demand for more efficient, cheaper and delivered on schedule software systems suggests that software development needs to take place in an environment that allows proved solutions to be modified, combined and adapted to be used in new software construction projects.

One of the answers provided by Software Engineering to this question can be found in the field of Domain Analysis, which can be defined as the study and organization of common aspects and variations existing among various software systems of an application domain. This process will provide a set of models describing the applications of a domain in a generic fashion, besides strategies to construct new software systems from the generic artifacts produced [4].

3.2 The FODA Method

During the late 1980s and early 1990s the Software Engineering community proposed several domain analysis methods. Despite the existing differences between them, these methods are functionally equivalent, invariably exhibiting operations such as aggregation, classification, specialization and parameterization [3].

Because of the existing similarities between these domain analysis methods, we have chosen the FODA method (Feature Oriented Domain Analysis) [4] developed by the Software Engineering Institute (SEI) as the method to be used in connection with our development approach. The choice of FODA can be credited to the vast documentation available, including some case studies, and the tight relationship between the models produced by the FODA method and those found in the majority of the Object Oriented Analysis and Design methods (OOADM). One of the main characteristics of FODA is the process of identification of the more relevant software features of an application domain. In the FODA method, a feature is defined as a user-visible aspect or a characteristic of a software system.

3.3 The Analysis of a Domain

In the FODA method, the domain model is subdivided into three distinct models that represent commonalties and existing differences between applications of a given domain. They are:

⇒ Feature Model – which captures the main user-visible aspects of systems in a domain.

⇒ Information Model – which captures the main abstractions, and the relationships between them, in an application domain.

⇒ Operational Model – which models the functional and behavioral aspects of applications in a domain.

The Feature Model is used to represent common features, and their relationships to each other, in a family of applications. Features are classified as mandatory, alternative and optional. Besides usual relationships, the model also captures structural relationships such as consist of, and generalization/specialization structures.

Compositional rules between features are also represented in the model. For instance, two alternative features that must simultaneously be present in an application of a domain have to be connected by the requires operator; while two other features that cannot simultaneously be present will be connected by the mutually exclusive with operator.

The Information Model is used to capture domain knowledge throughout the representation of its main abstractions and relationships. As with the Feature Model, aggregation and generalization/specialization structures will be added to the usual relationships between objects in a domain.

The Information Model can be represented by any tool capable of representing the semantic aspects of data, such as the Entity-Relationship Model [5] and the class diagrams of UML.

The objective of the Operational Model is to identify existing functional and behavioral variation between various applications of a domain. Tools like State Diagrams and Activity Diagrams found in UML can be used for this purpose. Nevertheless, such models have to be parameterized to properly represent existing variations.

4 The Reuse Model

The development of reuse artifacts has begun with the introduction of the extensibility characteristics into programming languages. It all begins by wrapping lines of code into a function/procedure element for later use like in Fortran. After that those procedure/functions could be stored into packages to be shared with other development departments. Object-oriented languages appear in this scene as an extreme reuse approach to software development. They claim that inheritance and composition of well-encapsulated specifications, called classes, will improve the development process by enabling designers to reuse these specifications as the starting point of their new software. This approach has evolved over time to the reuse of collection of related classes that can be domain specific, called Frameworks [6] or solutions specific called Design Patterns [14].

The problems with the Framework reuse can be stated as:

⇒ The designer should know the reuse artifact prior to its usage to be able to check if his problem (the application development) can take any benefit from reuse [18].

⇒ Once the application has been chosen, the designer must know what/how to reuse, that is, the reuse points (What are the extension/flexible points [1] to redefine? How to redefine?).

The first item is a problem-solution match that can be solved by a 'simple' program if designers use a formal description of both sides. Unfortunately these formal descriptions are hard to use, leading designers to use a more informal notation like UML. The problem with the UML notation is that only by reading plenty of informal descriptions, such as use-cases and attached notes, can the user capture the semantic of the design.

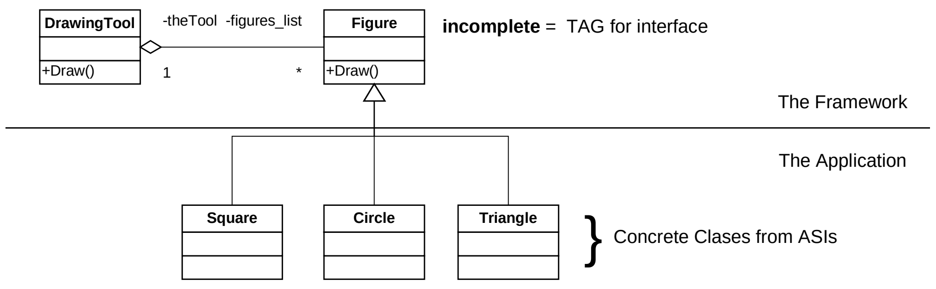

Figure 3 – DrawingTool Interface Extension hotspot example.

The second item reflects a documentation problem. To reuse a framework the reuser must be able to identify and define what Mattson called the application specific increments, that is, the code that is dependent on the application being developed that will be used to instantiate the framework. Here we have at least three problems:

⇒ Where to plug those application specific increments?

⇒ How to perform the plug?

⇒ What are the problems, and possible solutions, inherent to this plugging operation?

By definition, application specific increments are placed in hotspots, that is, in the framework flexible parts. In our work we use a subset of the Framework Design Language [17], FDL for short, for identifying the hotspots. We choose this language for two reasons. The first one is that with FDL we can express three kinds of extension points that, due to our framework development experience [9][19][16] and study of current solutions [18][7], can express hotspots in all dimensions. The three kinds of hotspots are variation methods, extension classes and extension interfaces.

The second reason for our choice is that FDL provides a complete mapping into UML description using stereotypes and tagged values. This is important because stereotypes and tagged values can be trapped in the UML XMI description to be manipulated by the environment.

Once we can identify the hotspots we need to know what to do with them, that is, we need to know how they can be instantiated. It is usual that during framework development, the framework designer instantiates a pilot application for testing purposes. During this instantiation the framework designer identifies the way to "fill" the hotspots with the application specific increments, from now on ASI. Usually, this filling action is expressed as basic O.O. instructions like methods redefinition or classes' specialization or as extended O.O. instructions such as design patterns application. Grouping these O.O. instructions, the framework designer defines what we call a framework reuse script that can be executed to guide framework reuse. Now our problems can be stated as the mapping of the three types of hotspots defined in the FDL to these O.O. instructions.

Extension interfaces are hotspots that need the definition of sub-classes to determine the ASIs and can be mapped to a basic class redefinition clause. In Figure 3 we use a generic drawing tool example where we have a class named Figure that was defined with the incomplete tagged-value. This means that to obtain ASIs the framework reuser must provide at least a class name for sub-classing.

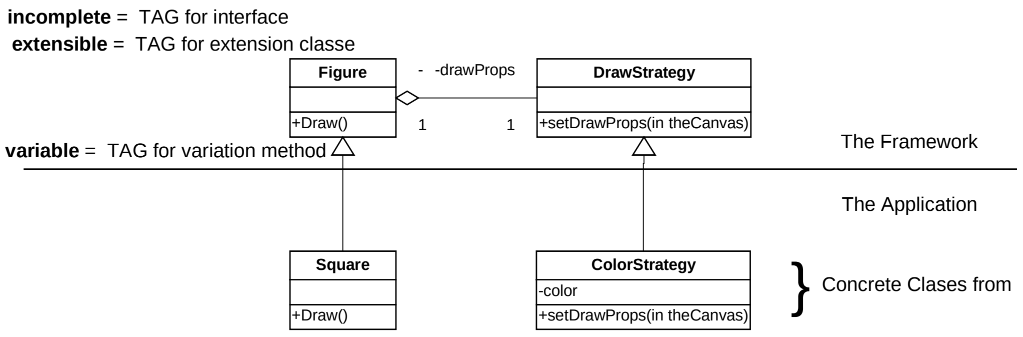
incomplete = TAG for interface
extensible = TAG for extension classe
Figure
+Draw()
- -drawProps
1
1
DrawStrategy
+setDrawProps(in theCanvas)
variable = TAG for variation method
The Framework
The Application
Square
+Draw()
ColorStrategy
-color
+setDrawProps(in theCanvas)
Concrete Clases from

Figure 4 – DrawingTool example for Extension Classes and Variation Methods.

Variation methods are methods that have a well-defined signature, but whose implementation may vary depending on the framework instantiation. This hotspot has no correspondence in basic O.O. instructions since methods code cannot be "reassigned." Using inheritance to override the variation method is not possible because the framework is based on the existing class. To solve this problem we use the Strategy Design Pattern [14] to transform this hotspot into a simple extension interface.

Extension classes are classes whose interfaces can be modified during instantiation. Here we have the same problem of variation methods, that is, traditional O.O.L. does not allow changes to this operation at compile time. The solution again relies on the strategy design pattern [14].

Using the same drawing tool example (Figure 4), suppose we need to add a color property to the Figure class. This instantiation changes the Draw method that needs the defined color. To achieve this ASI the Figure class must use the extensible tag to represent an Extension Class and the method Draw should use the variable tag to represent the Variation Method. To provide extensibility we use the Strategy Design Pattern so that the reuser can add the desired properties to the Figure class and use this property in the execution of the Draw method.

FDL makes a difference between ASI made at compile-time and run-time. Compile-time hotspots should be instantiated statically in code and are defined by the static tagged value. In contrast Run-time hotspots should be instantiated at run time through a reflective mechanism or any mechanism that allows changes into the class structure at run-time. Since our method is made for design composition it only deals with compile-time hotspots.

5 A Case Study

To provide a complete understanding of the approach, in this section we report about a fictitious case of a Rental System development that was the approach first use. The purpose of the Rental System is to rent vehicles to users, providing control over the transactions within the Rental Company. It is important to point out that during the domain analysis phase it was defined that the system matches the definition of workflow systems adopted by the WFFrame [19] framework, where a Rental could be represented as a workflow node with internal operations.

According to our approach the development begins with the creation of a Features Model [4], where we can represent the system's main characteristics. Figure 5 shows that payment is accepted at reservation time, rental time or return time. Beyond this, payment may be made by credit card, check or cash.

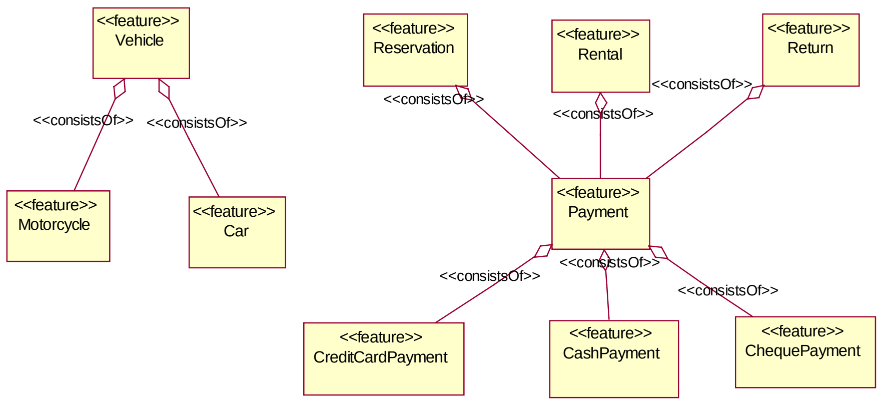

Figure 5 - Features Model for the Rental Example.

The next step is the construction of the Use Case Model. Most of the features can be traced to a use case. Optional features are represented as extension points in the Use Case Model [1]. For instance, a payment is an extension point in the Report Reservation of a Vehicle, Report Rental of a Vehicle, and Report Return of Vehicle use cases. Extension points are represented as tagged values attached to a use case through a UML note.

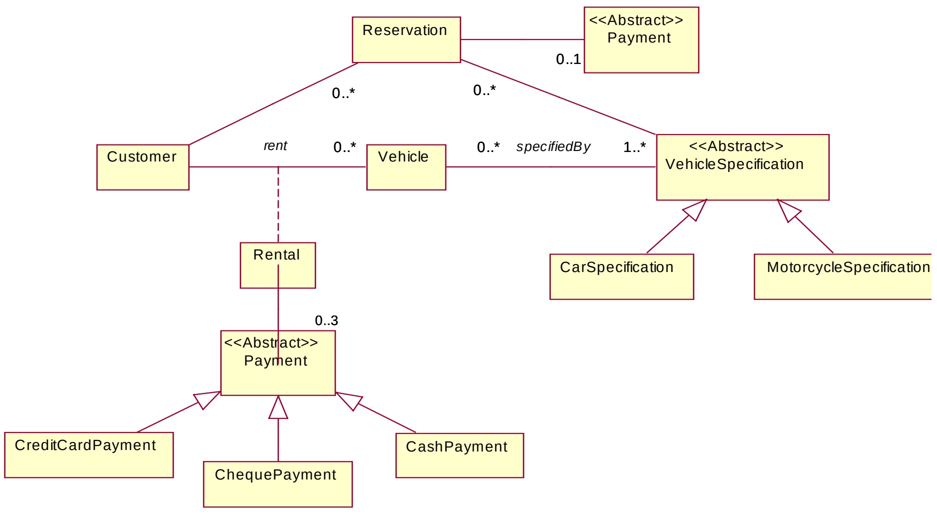

Figure 6 – Class Diagram for the initial specification of the Rental Example.

Both Use Case Model and Features Model are important. The objective of the Features Model is to provide a description of the main characteristics of the system. The Use Case diagram is a thorough description of the functional requirements of the system and will drive the development process [13]. The matching of these two diagrams will provide a way to check the integrity of the domain knowledge.

Once the application developer has satisfactory knowledge of the domain, it is time to represent this knowledge in a class diagram (Figure 6), which will be used as the basis of the reuse step.

The WFFrame is a framework in which we can define nodes that have associated components for presentation persistency and control. To provide ASIs the application developer must redefine the classes marked with the incomplete tagged value as shown in Figure 7. The reuse script defined for the WFFrame guides this redefinition step.

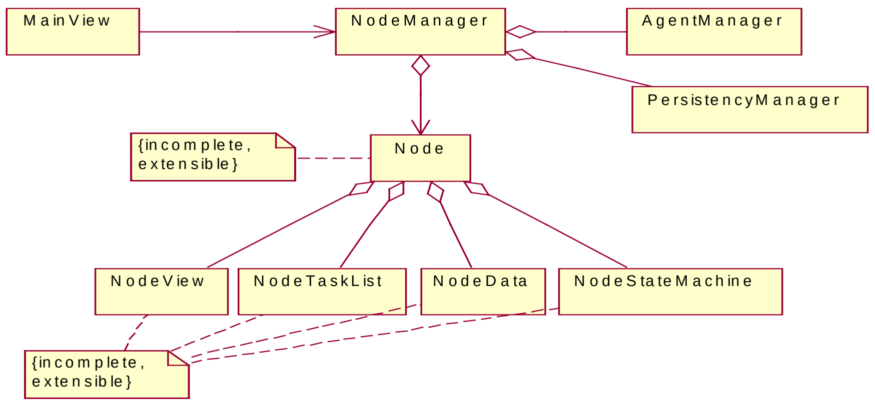

Figure 7 WFFrame Main Class Diagram

WFFrame Redefinition script:

⇒ Create Node Classes

⇒ For each new Node Class, Inherits from Class Node.

⇒ For each new Node Class, create View classes

⇒ For each new View class, inherits from Class NodeView

⇒ For each new Node Class, create Data class

⇒ For each new Data class, inherits from Class NodeData

⇒ For each new class Node, Redefine Method CreateData with Abstract Factory[14]

⇒ For each new NodeData class Redefine Method Load

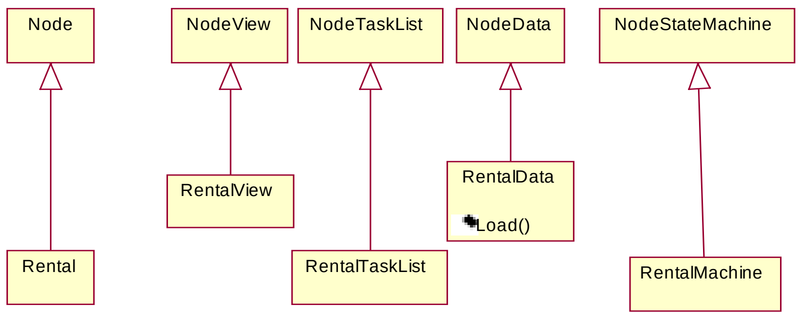

Figure 8 – Rental example redefinition based on the Rental node.

The execution of this script takes as input the XMI specification that is obtained from both the Rental Design and the WFFrame Design, which in this case was created with Rational Rose Case Tool and the IBM XMI Toolkit. Once executed this script generates a class diagram that will provide the design of the whole application (Figure 8).

6 Conclusion & Future Works

Brazil’s Ministry of Defense used the approach presented in this paper with great success for the development of three real applications. The development was carried out using the Borland Delphi programming environment and the focus domain was Decision Support Systems. During the development of the second application the domain knowledge was completely obtained and the cost and time reductions were used as the basis of the third system schedule. The whole system can be measured as a 55.000 LOC system, from where about 23.000 was obtained from reuse.

Due to the simplicity of the approach presented in this paper and due to our development experience we believe that it can be used in a great variety of Information Systems where top-down decomposition in the design is a largely used technique.

Although the approach was successful in its first use, a lot of improvements must be made and we divide them into two groups. The domain modeling approach and the reuse approach.

In the domain-modeling phase we need to create annotations in the models to be able to capture things such as inheritance and generalization to improve the mapping from the XMI to the O.O. design model. In the reuse model we need to be able to express more redefinition points that are not defined as hotspots or frozen-spots in the frameworks design but, rather, as an intermediary "spot" where the designer is not obliged to define ASI. This is particularly important when the framework provides user interfaces that "can" be customized.

Another important extension is the construction of an integrated environment to support the execution of the reuse script, using a kind of wizard to avoid name conflict and multiple inheritance conflict using Refactoring [15] in the application design.

7 Bibliography

[1] Jacobson, I.; Griss, M.; Jonsson, P. *Software Reuse: Architecture, Process and Organization for Business Success*. Addison-Wesley, Reading, Massachusetts, June 1997.

[2] Argo/UML Description found at http://www.argouml.org.

[3] Arango, G.. *Domain Analysis Methods*. In Advances in Software Reuse: Selected Papers from the Second International Workshop on Software Reusability, p.17-49, March 1993, Lucca, Italy. Edited by Ruben Prieto-Diaz and William B. Frakes, IEEE Computer Society Press, 1993.

[4] Kang, K.C.; Cohen, S.G.; Hess, J.A.; Novak, W.E. and Peterson, A.S.. *Feature-Oriented Domain Analysis (FODA) Feasibility Study (CMU/SEI-90-TR-21)*. Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University, Nov 1993.

[5] Navathe, S.B.; Batini, C.; Ceri, S.. *Conceptual Database Design – an Entity-Relationship Approach*. Benjamin Cummings, Redwood City, California, 1992.

[6] Fayad, M.E., Schmidt, D.C., Johnson, R., Implementing Application Frameworks, Wiley 1999.

[7] Pree W. *Design Patterns for Object-Oriented Software Development.* Addison-Wesley, Reading Mass., 1994.

[8] Silva, A.R. *Programação Concorrente com Objetos: Separação e Composição de Facetas com Padrões de Desenho, Linguagem de Padrões e Moldura de Objetos*. Dissertação de Doutorado Universidade Técnica de Lisboa – Instituto Superior Técnico Portugal 1999.

[9] Oliveira, T.C. ; Carvalho, S.E.R. ;Lucena C.J. P. *DSSFrame - A Decision Suppot System with Agents*. Techinical Report Pontifícia Universidade Católica do Rio de Janeiro – Brazil 2000.

[10] XMI Specification found at http://www.omg.org/technology/xml/index.htm.

[11] Pressman, R.S. *Software Engineering : A Practitioner's Approach*. McGraw Hill, New York, NY, June 2000.

[12] Sommerville, I. *Software Engineering*. Addison-Wesley, Reading, Massachusetts, August 2000.

[13] Jacobson, I. *The Unified Software Development Process.* Addison-Wesley, Reading, Massachusetts, February 1999.

[14] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, Massachusetts, October 1995.

[15] Fowler, M. *Refactoring : Improving the Design of Existing Code*. Addison-Wesley, , Reading, Massachusetts, June 1999.

[16] Fontoura, M.; Crespo, S.; Lucena, C.J.P.; Alencar, P.S.C.; Cowan, D.D. *Using Viewpoints to Derive Object-oriented Frameworks: a Case Study in the Web-based Education Domain*. The Journal of Systems and Software, 54 (2000) 239-257

[17] Fontoura, M. F. M. C. *A systematic approach to framework development*, PhD Thesis, Department of Computer Science, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), 1999.

[18] Mattsson, M. *Evolution and Composition of Object-Oriented Frameworks*, PhD Thesis, Department of Software Engineering and Computer Science, University of Karlskrona/Ronneby, 2000.

[19] Oliveira T. C., Mathias I., Lucena, C. J. P. *A Framework Approach for Workflow Software Development* Proceedings of IASTED International Conference on Software Engineering and Application, p330-335, Las Vegas USA, November 2000.