Domain Oriented Framework Construction

Ivan Mathias Filho
Department of Computer Science, Pontifícia Universidade Católica do Rio de Janeiro, Rua Marquês de São Vicente 225, Rio de Janeiro, RJ, 22453-900, Brazil
Email: ivan@inf.puc-rio.br

Toacy Cavalcante de Oliveira
Department of Computer Science, Pontifícia Universidade Católica do Rio de Janeiro, Rua Marquês de São Vicente 225, Rio de Janeiro, RJ, 22453-900, Brazil
Email: toacy@inf.puc-rio.br

Carlos J.P. de Lucena
Department of Computer Science, Pontifícia Universidade Católica do Rio de Janeiro, Rua Marquês de São Vicente 225, Rio de Janeiro, RJ, 22453-900, Brazil
Email: lucena@inf.puc-rio.br

PUC-RioInf.MCC23/01 July, 2001

Abstract: Object-oriented Application Frameworks is a powerful reuse technique that allows the sharing of requirements, design and code among a set of application systems instantiated from an original framework. Nevertheless, little attention has been given to the role of an explicit Domain Analysis phase in the framework construction process. This paper describes an approach where the requirements capture for an entire application family takes a central role in the framework development, thus facilitating verification processes and providing reliable documentation that will assist the instantiation step.

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

Resumo: Os frameworks orientados a objetos constituem-se em uma poderosa técnica de reutilização que permite o compartilhamento de requisitos, design e código entre um grupo de aplicações instanciadas a partir de um mesmo framework. Entretanto, pouca atenção tem sido dada à inclusão de uma etapa explícita de Análise de Domínio no processo de construção de frameworks. O presente trabalho descreve uma abordagem onde a etapa de elucidação de requisitos para toda uma família de aplicações assume um papel central no desenvolvimento de frameworks, tornando mais fácil o processo de verificação e fornecendo documentação confiável que irá auxiliar na etapa de instanciação.

Palavras-chave: Reutilização, Framework, Análise de Domínio, UML, XML, Orientação a Objetos

1 Introduction

The software development process can be seen as a set of steps that produce models of a decreasing level of abstraction up to the code, its ultimate product. These different levels of abstraction bring out what we have called an inter-phase semantic gap, which makes the transition from one phase to the next more difficult. Moreover, in the early stages of development, knowledge is fragmented and incomplete so software engineers have to use a mix of free text and visual models to capture stakeholders' needs.

In the field of framework construction this problem is amplified because the requirements to be collected are not for one application but for an entire family of applications [1]. The design of a framework has to be flexible enough to generate instances, members of the family, in accordance with its own requirements, and to accommodate future needs.

This paper describes an approach that tries to minimize the semantic gap through a series of refinement steps. It begins by making a thorough domain analysis to determine the common and different aspects of an application family with the FODA Method [5] and Use-cases [9], and proceeds in an iterative way [9] until the framework design is accomplished. The framework design is then expressed in a precise notation that clearly separates the hot-spots from the frozen-spots [2].

The models produced during the full process are tied together through adoption of industry standard languages like UML and XMI, which are used as the basis of the representation of the diagrams involved. This common background representation will be of crucial importance for the construction of verification tools that will be used to maintain the consistency among the models and for documentation purposes.

Section 2 describes the approach overview. In Section 3 we describe how domain information is collected. In Section 4 the process for framework construction 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 usual approaches for framework construction do not give the proper support to the requirement elicitation phase of the development process. The consequences are the absence of mechanisms that allows the developers to identify the origins of the hot-spots and the frozen-spots of a framework, and the absence of tools for validation of a framework instance, regarding its set of functional requirements in relation to the set of functional requirements of the application family that originated the framework.

So, our approach begins with the Requirement Analysis Phase, where the functional and non-functional system requirements are collected to represent its main functionality (from the end user's point of view). After that the framework designer moves to the Design Phase

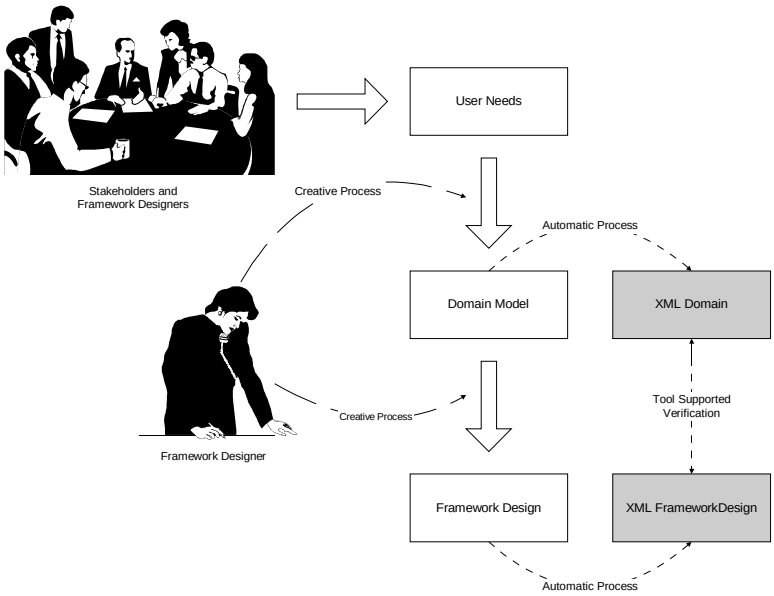

Figure 1 – The approach overview.

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 system functionality. Usually, the designer goes back to the first phase to begin another development cycle and to reach the desired level of application representation (Figure 1).

To represent the framework 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 framework semantics and provide compatibility with market standards. Since our approach combines O.O. designs, we are particularly interested in the class diagrams and how to obtain them. Naturally, class diagrams are a structural refinement of domain designs that in our approach are obtained by combining an initial conceptual model, Features Diagrams, and Use-cases in a guided way, to provide traceability. The next step is to represent the dynamic aspects of a domain. This will be done by constructing a set of collaborations that realize the domain use-cases, assigning responsibilities to the classes of the domain. 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. In order to do this, they propose a complete representation of UML using the XML format that can be shared among software vendors. The XMI representation of a design can be achieved using case tools like Argo/UML or using the IBM XMI toolkit converter for Rational Rose.

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 [3] and Domain Design [4]. With this approach we tend to isolate the application domain design from the techniques used for framework instantiation, such as inheritance, parameterization and design patterns.

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 proven solutions to be modified, combined and adapted for use in new software construction projects [3].

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 the

common aspects and variations that exist among various software systems of an application domain. This process will provide a set of models that describe the applications of a domain in a generic fashion, besides the strategies used to construct new software systems from the generic artifacts that are produced [5].

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 [4].

Because of existing similarities between these domain analysis methods, we have chosen the FODA method (Feature Oriented Domain Analysis) [5], 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 [11][13], 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 the 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 also are represented in the model. For instance, two alternatives 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.

Any tool capable of representing the semantic aspects of data, such as the Entity-Relationship Model [6] and the class diagrams of UML can represent the Information Model.

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

3.4 Applying Domain Analysis to the Proposed Approach

The objective of this section is to explain how Domain Analysis techniques, especially the FODA method, will be integrated into our framework development approach. The first question that arises is why did we choose the FODA method? First, because it represents the domain knowledge using several complementary models. Second, because its most interesting model, the Features Model, matches well with use-cases, which is important since our approach is object-oriented.

The initial step in developing software systems is to identify the needs of the users and other stakeholders. These individuals have business or technical problems that require our help (software developers) to solve them. So it is our responsibility to understand the problem

domain and translate the stakeholders' needs into a set of services, called features, that the system will provide to address those needs [12].

The features address one or more stakeholders' needs and reside in the solution domain, the land of computers, programming, operating systems, etc. It is beyond the scope of this work to specify how the features are obtained. What concerns us is that once the list of features is available we will structure it like the FODA's Features Model.

The second step is to define more specific requirements that conform to stakeholder needs. These more specific requirements are the software requirements; that is, those things that the software does on behalf of the players who interact with the system.

At this point it is important to distinguish Features from Software Requirements. While the Features Model captures the desired functionalities provided by the system in a high-level abstraction, the Software Requirements should express these features in a more detailed fashion. That is, the Software Requirement artifacts have to be of a level of abstraction that allows software engineers to lay down the initial design. Nevertheless, despite the differences between the two models there is a mapping that relates features to software requirements (this is explored in more detail in the next section).

3.5 The Relationship between Features, Use-cases and other Artifacts

The Features Model and the Information Model, represented here by a Conceptual Model or a first-cut Class Diagram, constitute the Domain Model of our approach, and is the basis from which the Domain Use-case Model will be generated [9]. The Domain Use-cases Model expresses the functional requirements of an entire application family, and will play the main role in the Software Requirements Specification [9][12]. Although non-functional requirements are beyond the scope of this work, they can be attached to the use-cases that they concern or kept in a separate document (Supplementary Requirements) [9]. Figure 2 illustrates the process of obtaining the Software Requirements Specifications from the Domain Model (Feature Model plus Information Model).

Some features can be associated with use-case or use-case sections, while other features can be associated with the type of players, virtual operations, parameters in objects, target

platform, etc.[13]. Although one feature can be related to only one use-case, use-cases can be traced back to many features.

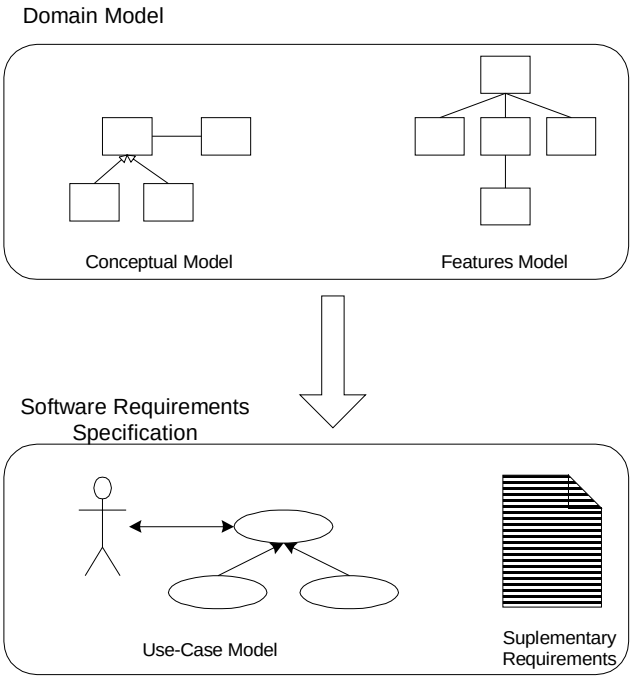

Figure 2 – Obtaining Software Requirements Specifications from the Domain Model.

As stated previously, the objective of an explicit Domain Analysis phase is to capture the common and variable aspects of an application family. Alternative features represent variability that occurs at variation points [14]. There are many kind of variability mechanisms used to represent variations. Inheritance is the mechanism used when the variation point is a virtual operation, and extensions are the mechanisms used when the variation point is an extension point. When an extension point is located in a use-case, the extensions are others use-cases that are linked to the former use-case through the <<extends>> relationship.

So, some features -- the ones that better expresses alternative functionalities -- will be mapped to extension use-cases (linked to the more abstract use-case through <<extends>> relationships), while other types of alternative features will be mapped to models that can capture abstract structures and abstract behaviors (abstract classes, abstract operations, etc).

Now that the roles of the Features and the use-cases (two closed related concepts) are well established, another question has to be answered: are the Features Model and the Use-Case Model both necessary? The answer is yes, because while the Features Model captures the structural relationships between the features in an easy to understand and concise way, the Use-Case Model describes a subset of those features as a set of actions that will be used later to realize the features as interactions between objects of the Design Model; that is, the Object-Oriented Design of an Application Family or, as it will be seen in a later section, an Object-Oriented Application Framework.

3.6 The Role of Traceability

Traceability is an effective technique that help us understand the relationships between the objects (features, players, use-case, classes, etc.) of our projects, thus permitting verification activity. The verification process helps us maintain the consistency among the models and to discover things that have been missed or are superfluous. Supported by the right tools, traceability permits us, for example, to check if an alternative feature has been mapped to an extension use-case or virtual operation, or if a hot-spot doesn't trace back to any alternative or optional requirement, which would be an error.

Traceability is divided into two categories: implicit and explicit [12]. An implicit traceability may arise naturally from the modeling paradigm and the associated modeling language used. For example, an implicit trace relationship occurs in UML when a player communicates with a use-case or a subclass inherits from its super-class.

On the other hand we define explicit traceability as the development of a relationship that is derived from external consideration supplied by the developers [12]. So, for instance, there is no intrinsic relationship between a feature and a use-case. If such a connection is established, it is based only on the semantic of the constructions.

It is important to state that the use of XML/XMI as the target design description language will reduce significantly our effort in building tools to support traceability. There are many XML tools on the Internet that can be adapted for such a purpose.

4 Object-Oriented Application Frameworks

4.1 Frameworks as a Reuse Technology

Object-oriented Application Frameworks, hereinafter called frameworks, are an object-oriented reuse technique based on other object-oriented techniques, such as inheritance, polymorphism and design patterns. Frameworks provide a built-in architecture and a set of incomplete methods representing an abstract collaboration model or a pattern of object interaction.

Frameworks can be defined as "a reusable design of all or part of a system that is represented by a set of abstract and concrete classes and the way their instances interact." [7]. Another good definition states that a framework is "a system composed of a kernel subsystem, which is common to all applications that may be generated within the framework, and a hot-spot subsystem, which implements the application-specific behavior." [15]. No matter which definition we choose, one central idea is always present: the reuse of software requirements and complex designs.

By definition, application specific increments, from now on called ASI, are placed in hot-spots, that is, in the framework flexible parts. In our work we use a subset of the Framework Design Language [15], FDL for short, for identifying the hot-spots. We have chosen 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 [8] and study of current solutions [2], can express hot-spots in all dimensions. The three kinds of hot-spots 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.

Extension interfaces are hot-spots that need the definition of sub-classes to determine the ASIs and can be mapped to a basic class redefinition clause. This means that to obtain ASIs, the framework reuser must provide at least a class name for sub-classing.

Variation methods are methods that have a well-defined signature, but whose implementation may vary depending on the framework instantiation. This hot-spot has no correspondence in basic O.O. instructions, since methods can not have its code “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 [10] to transform this hot-spot 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 when it is time to compile. The solution again relies on the strategy design pattern [10].

FDL distinguishes between ASI made at compile-time and run-time. Compile-time hot-spots should be instantiated statically in code and are defined by the static tagged value. In contrast Run-time hot-spots should be instantiated at run time through a reflective mechanism or any mechanism that allows changes into the class structure at run-time.

4.2 Deriving Frameworks from Domain Models

Instantiating a framework is a complex process that does not only deal with the specialization of a generic design. Beyond this, the instantiation process has to deal with the specialization of a generic set of software requirements, those that were specified for an entire application family. So the framework development process has to take into account, as of the early stages of the development, the artifacts produced by the Domain Analysis phase.

The static structure of a framework that is represented by Class Diagrams will be directly derived from the Conceptual Model plus the addition of other design classes. The pattern of interaction among the objects of the framework will be based on the interaction diagrams that realize the Domain Use-cases. Since the Domain Model artifacts are naturally abstracts of the Design Model, the framework obtained by such a process will be equally abstract. The mandatory features will be mapped to the frozen-spots while the variant features, and some optional features, will be reflected in the hot-spots in some way.

Due to the documentation problem stated in Section 4.1, it is very important that during the instantiation of a framework the developers can trace back to the software requirements that

the framework is based upon. Moreover, the application that will be generated has to conform with some of the alternative features and has to include some of the optional features. This process has to be ideally supported by verification tools that can report the existence of inconsistencies in the instantiated application to the developers.

Now the role of the FDL introduced in the previous section became clearer. The FDL precisely and explicitly represents the existence of a hot-spot and its nature. Since the FDL and the Features Model are compatible with each other and are described by the same language -- for example XMI -- the construction of such verification tools will not be a difficult task.

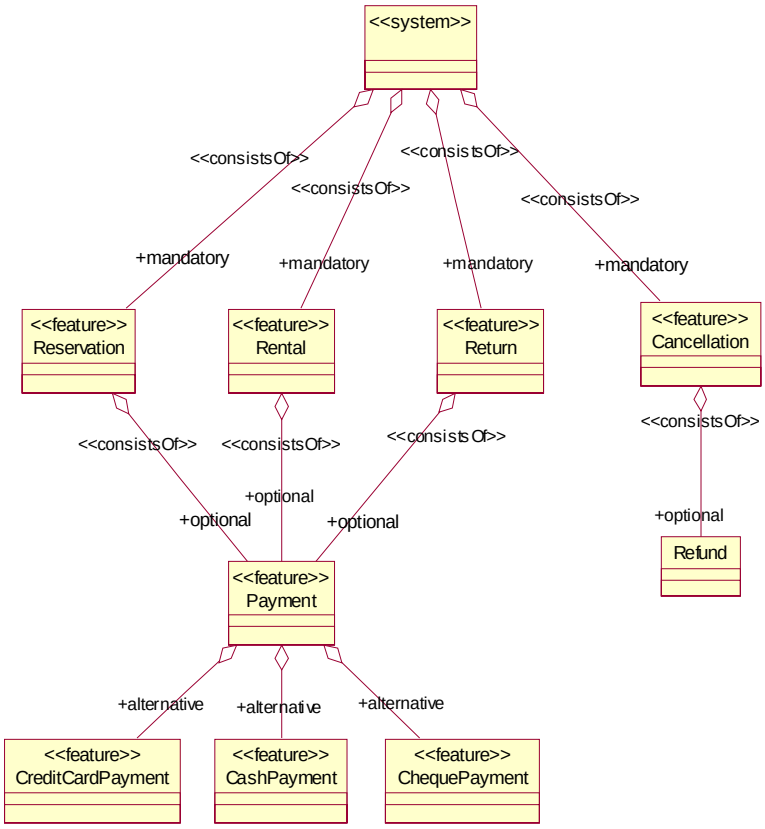

Figure 3 – Features diagram of the Rental System.

The Case Study introduced in the next section will clarify some of the ideas that were presented so far.

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 that was first used. The purpose of theRental System is to rent vehicles to users, providing control over the Rental Company's transactions.

According to our approach, the development begins with the creation of a Features Model [5] where we can represent the system's main characteristics. Figure 3 shows that a payment is accepted at reservation time, rental time or return time. Moreover, credit cards, checks or cash can be used for payment purposes.

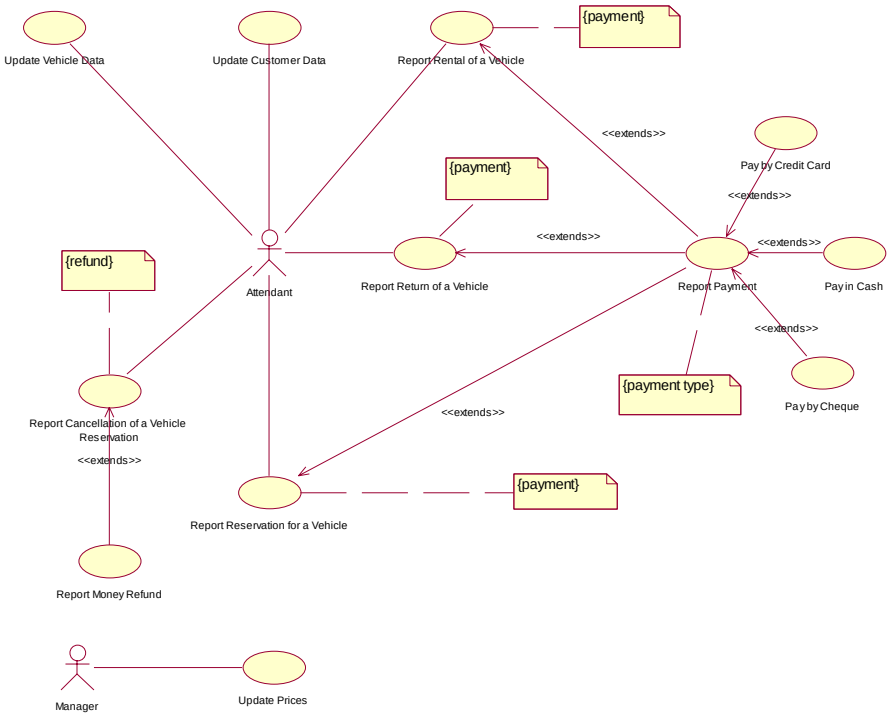

Figure 4 – Use-case diagram of the Rental System.

The next step is the construction of the Use-case Model (Figure 4). Some functional features trace to whole use-cases while other trace to use-case sections. We can associate extension points with use-cases to represent alternative and optional features in the Use-case Model. 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. Those extension points reflect the fact that the features Reservation, Rental and Return are composed of an optional feature, called Payment.

Another extension point, called payment type, is associated with the Report Payment use-case. This extension point is used to represent the three alternative ways of making a payment: through credit cards, checks or cash. All of the three alternatives are represented as alternative features in the Features Model, and are traced back to the Payment feature through the consistOf relationship. An extension point is represented as tagged value attached to a use-case through a UML note.

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 [9]. The matching of these two diagrams will provide a way to check the integrity of the domain knowledge.

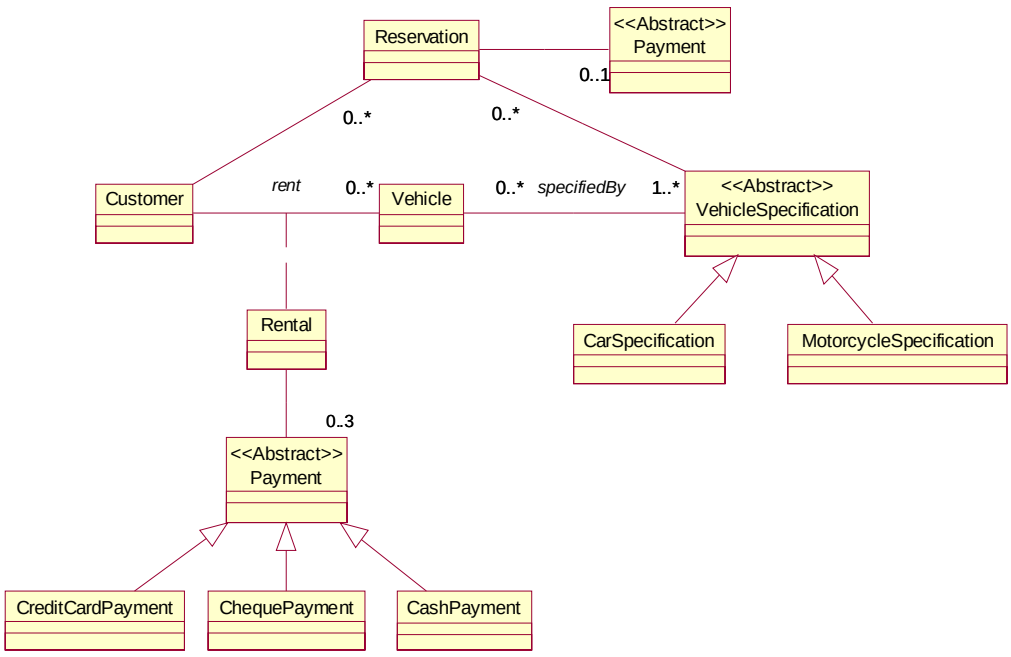

Figure 5 – The initial conceptual model.

Once the application developer has a satisfactory knowledge of the domain it is time to represent this knowledge in a initial Conceptual Model (Figure 5), represented as a class diagram.

The Conceptual Model will evolve trough the addition of operations and design classes, and will further represent the static view of the framework. The hot-spots are represented as tagged values attached to the classes through UML notes. For example, the fact that the types of vehicles for rent can vary for each framework instance is represented as an Interface Hot-spot.

Although the Class Model is the main model of a framework, it cannot capture all of the variation details embedded in the framework. For example, to better explain the fact that a payment is an optional feature in a reservation process we can construct a sequence diagram that can clarify our understanding. Figure 6 shows such a diagram.

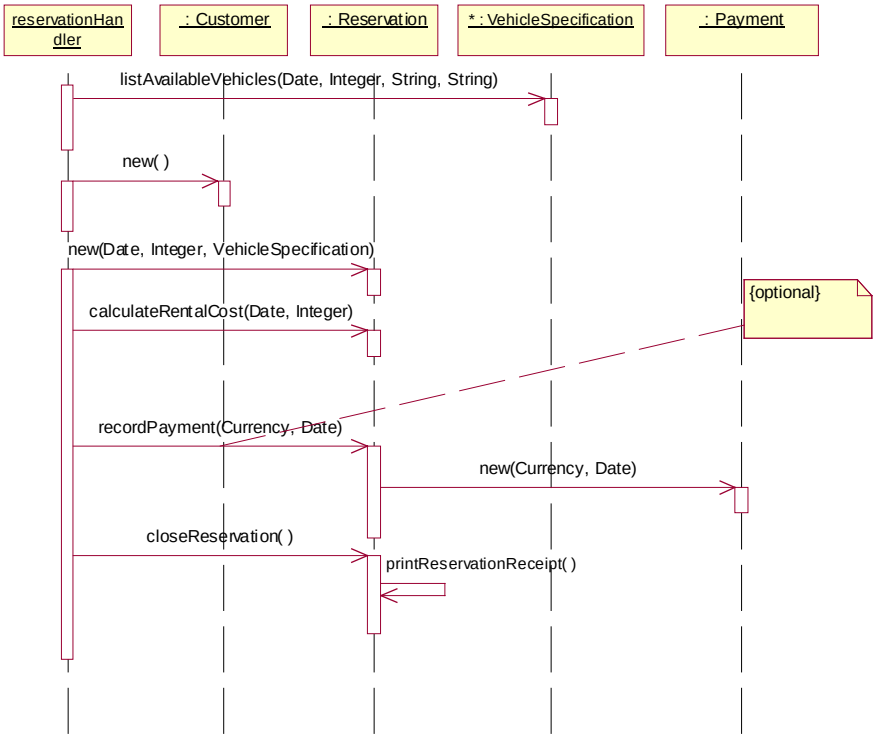

Figure 6 – An optional feature represented as an optional operation call.

6 Conclusion & Future Works

This paper describes a thorough and well structured approach to construct frameworks based on domain models. Although the proposed method works well when applied to Information Systems, it is necessary to apply it to other types of systems, especially ones that are not based on user interfaces, to verify if it is a general approach.

Despite its strengths, a lot of improvements need to be carried out and some tools must be constructed. It is necessary to create a mechanism to support the Features Model in the UML. Although the stereotype mechanism seems to be useful at first sight, perhaps we have to extend the UML Metamodel to accommodate the Features Model. Moreover, we need to be able to express more redefinition points that are not defined as hot-pots or frozen-spots in the frameworks design but rather as an intermediary spot so that the designer is not obliged to define ASI. This is particularly important when the framework provides user interfaces that can be customized. It is also necessary to improve the mechanism used to express the FDL constructions in UML since, for example, it is not possible to use a UML note to attach a tagged value to a specific class operation

Another important extension is the construction of a set of tools to support the verification process based on trace relationships. Such tools will take advantage of the fact that all of the models are represented in XML/XMI.

7 Bibliography

[1] Parnas, D.L. *On the Design and Development on Program Families*. IEEE Transaction on Software Engineering, Vol. SE-2, No. 1, March 1976.

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

[3] Prieto-Díaz, R.. Historical Overview. In Advances in Software Reuse: Selected Papers from the Second International Workshop on Software Reusability, p.1-16, March 1993, Lucca, Italy. Edited by Ruben Prieto-Diaz and William B. Frakes, IEEE Computer Society Press, 1993.

[4] 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.

[5] 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.

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

[7] Fayad, M.E., Schmidt, D.C., Johnson, R. Building Application Frameworks, John Wiley, New York, NY, 1999.

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

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

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

[11] Vici, A.; Argentieri, N.; Mansour, A.; Favaro, J.; d´Alessandro M. FODAcom: An Experience with Domain Analysis in the Italian Telecom Industry. 5th International Conference on Software Reuse (ICSR-5), ACM/IEEE, Victoria, Canada, June 1988.

[12] Leffingwell, D.; Widrig D. Managing Software Requirements – A Unified Approach. Addison-Wesley, Reading, Massachusetts, 2000.

[13] Griss, M.; Favaro, J.; d´Alessandro M. Integrating Feature Modeling with the RSEB. 5th International Conference on Software Reuse (ICSR-5), ACM/IEEE, Victoria, Canada, June 1988.

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

[15] 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.