A Framework Development Process for Product-Line Architectures

Detlef Schmitt

Sema GmbH, Kurt-Schumacher-Strasse 241, 26389 Wilhelmshaven, Germany
and
Open University, Milton Keynes, MK7 6AA, United Kingdom

Phone: +49-4421-802-224, Fax: +49-4421-802-555,
Email: Detlef.Schmitt@so.sema.de

Abstract. This article describes the background, objectives, processes, and artifacts of an industrial approach to the design and construction of a product-line architecture for discrete event simulation by using a small framework as its kernel. It will demonstrate how the concepts of software product-lines and object-oriented frameworks were combined in order to establish a sound basis for developing a whole family of similar programs.
After introducing the reader to the general situation and conditions of the project, the whole process for domain analysis, solution design, construction, and implementation of the framework is explained and illustrated in some detail. The article finishes with the lessons learned and a summary of the experience gained in the process.

1 Introduction

Software Product-Lines became known during recent years as a way to design families of related software systems (e.g. [Weiss 1999] and [Bosch 2000]), although their concepts reach back to the seminal work of David Lorge Parnas [Parnas 1976]. The logic behind this thinking is the economy of reuse by building similar applications that are all based on a set of adaptable software components. These products are constructed to conserve the knowledge of the experts in a given domain. They satisfy the more common requirements of the domain and enable the adaptation and reuse of already existing software designs as well as their associated program code packages. A further goal is to reuse software architectures that have matured during prior application.

The aim of software product-line usage is not only cost reduction, but it is also the quest for higher quality, shorter time to market, simplification of application software development, and expected gains in productivity.

During the last decade, object-oriented frameworks too became more and more common. They are now in wide-spread use and are available for domains like graphical user interfaces (GUI), mathematical calculations, manufacturing, financial applications, banking, and many more. Fayad et al. describe many different frameworks in detail [Fayad 1999]. The main advantage of framework application and usage is the

ability to enact reuse in a way that is more economic than it was possible before this technology's introduction. An object-oriented framework can serve as the kernel of a software product-line satisfying the general requirements of that domain for many different implementations.

Software development teams at the author's company had implemented many slightly differing simulation applications before, so it was consequential to look for an approach to develop such programs more efficiently and evolve these products subsequently into a complete software product-line.

Current object-oriented analysis and design methods and their associated development processes (e.g. described in [Booch 1994], [Jacobson 1999], or [Kruchten 2000]) are mainly concerned with the creation of new applictaions and are not focussed on the reuse of existing artefacts. Therefore the main objective of this paper is to show how a systematic development process for object-oriented frameworks was applied in an industrial approach to software construction of a simulator product-line based in the domain of discrete event simulation. Finally the paper draws the conclusions from the application of the process and summarizes the lessons learned.

2 Background and Goals of the Project

2.1 Background and Settings

For many years, the author's company had produced simulators in the area of real-world physical object simulations like air traffic control, surveillance and search radars, fire-control subsystems, naval command and control applications, and many more. The simulators are needed for two main purposes:

- To supply a replacement for external hardware or software which was either not yet available for playing the role of the real external subsystem, too costly or too risky in its usage.
- To reduce cost and risk in using simulators instead of parts of the real system.

A typical simulator normally comprises the following components:

- A simulator control, which is responsible for the preparation, management, control, and conduct of the simulation runs,
- a display component taking care for the simulator's human-machine interface as far as it is concerned with preparing, running, and controling the scenario,
- a property control which is in charge of the simulated entities' properties, and
- an interface to the system under development, for test, or training.

All the simulators constructed in the past were individually built and handcrafted. Although the expertise and knowledge accumulated in all the products built before was inherently reused by the developers, only opportunistic code reuse took place [Bosch 2000]. No preplanned reuse process was established, which led to a repeated waste of resources. Since all simulators were similar and based on the concepts of discrete event simulation, the idea to establish a software product-line was born.

2.2 Goals and Objectives

When starting to think about a software product-line, a number of goals and objectives were identified:

- The resulting product-line architecture had to be flexible, in order to be adapted to different requirements for the simulators to be built.
- The software quality had to be enhanced based on the reuse of already existing and tested artifacts.
- The cost for development of a new individual simulator had to be reduced by reusing already existing, tried, and tested components.
- All software components were required to be as simple, abstract, and configurable as possible.

The next section describes the process for the development of one of the software elements for the simulator product-line - in this case the functionality of the basic simulation - which was developed as an object-oriented framework. The other components of the product-line were constructed in a more conventional way, although the display component and the property control are using the commercial framework Microsoft Foundations Classes (MFC) as their basis. The functionality of these two components is packaged in a black-box approach using the Microsoft ActiveX technology. This part of the product-line will not be described any further in this paper.

3 A Process for the Development of an Object-Oriented Framework

3.1 Overview of the Process

Generally, a framework has to be developed in an iterative, incremental, and adaptive process. The main reason for this is the fact that it is not possible to know the final structure and functionality of the framework from the outset. Knowledge and insight discovered later or in more concrete phases of the process will influence earlier or

more abstract activities, results, and products. This leads to the necessity to work on different levels of abstraction in parallel. Even if the whole process is illustrated in sequential steps due to the inherent chronological order of a research paper, the method is actually iterative and incremental.

The development process for the framework starts with a thorough domain analysis. This activity is based on expert knowledge, prior experience with comparable program systems, the analysis of already existing and running applications of similar kind, and eventually a prototype built for experimenting with the parameters determining the simulator system.

In the next phase, the framework is designed, i.e. the classes found in the domain analysis describing the problem are mapped to the classes of the solution space and are refined together with their attributes, operations, and mutual relationships.

The framework is then implemented in an object-oriented programming language suitable for this task based on the framework design model and the related documentation.

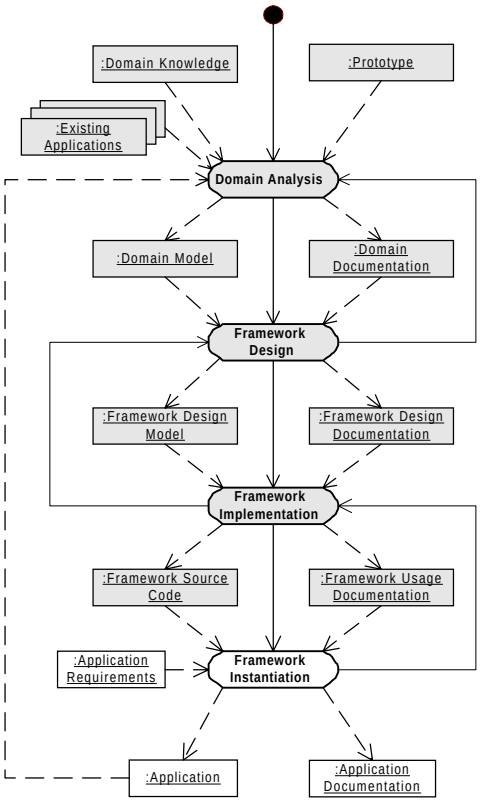


Figure 1: High-Level View of the Framework Development Process

In a last step the framework has to be instantiated, i.e. adapted by the application programmer on the basis of the requirements for the application. The results are a completed application and the documentation supporting it.

A high-level and simplified representation of the whole process in the form of an activity diagram as defined in the Unified Modeling Language ([Rumbaugh 1999] and [Boch 1999]) is given in Figure 1. In this figure the tasks of the development process are shown as activities, e.g. Domain Analysis, Framework Design, and so on. The products, e.g. the domain model and the domain documentation, are represented as objects and bound to the activities via dependency relationships (directed dashed lines with arrows). The transitions (i.e. the activity flows) connect the activities as directed lines with arrows.

To illustrate the incremental and iterative character of the whole process, feedback flows are shown as transitions between the tasks (activities) of the process.

The activities and products shown with a white background (i.e. framework instantiation, application requirements, application documentation, and the application itself) are not directly part of the framework development process. They are shown here for completeness and better understanding.

3.2 Domain Analysis of Discrete Event Simulation

The purpose of domain analysis is to capture and structure the information, knowledge, and expertise in a given application area. In order to solve the typical and recurring problems of that area in software, the reuse of the accumulated knowledge in the domain is a prerequisite.

Prieto-Diaz and Arango put it in their own words: "Domain analysis deals with the identification, acquisition, representation of (potentially reusable) software-specification and software-implementation knowledge for classes of real-world problems" [Prieto-Diaz 1991].

Discrete event simulation imitates real-world entities. These real-world entities change their behavior and state in time and through the influence of external events, also called triggers. Only a few basic entities which have to be modelled exist in this simulation concept which forms the basis of the domain being analyzed.

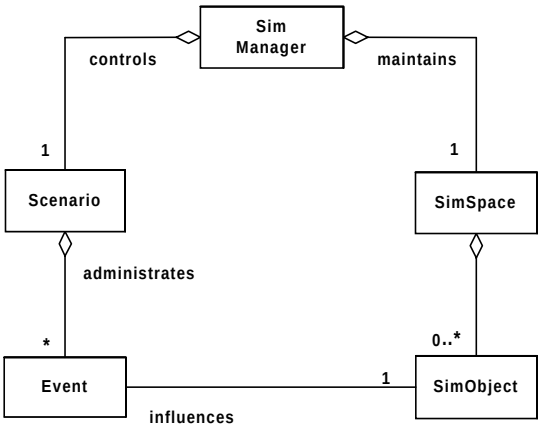


Figure 2: Class Model of the Simulation Domain

The CRC (Class - Responsibility - Cooperation) cards approach described by Beck and Wirfs-Brock et al. [Beck 1989, Wirfs-Brock 1990] was used to find the relevant classes in question. These classes represent the typical real-world or conceptual entities of the problem domain, their behavior, and relationships.

The main product of this phase is the domain model in form of use case, class, and sequence diagrams and the accompanying documentation. The use case diagrams and descriptions summon the requirements, the stakeholders - called actors - and the boundaries of the domain. The class diagrams show the relevant classes of the domain

under consideration and their mutual relationships. In figure 2 the highest-level class diagram as one products developed during the domain analysis is shown.

Where appropriate, sequence diagrams are used for describing interaction mechanisms between the classes found. If suitable, state transition diagrams depicting dynamical class behavior are introduced. Another part of the domain analysis is the study of scope, commonality, and variability of the problem domains' functionality and behavior [Coplien 1999]. In this part of the analysis it is decided which classes and elements of their behaviour are inside the domain scope, have common attributes, and variable elements. In close conjunction with the framework design and in an iterative approach, the bais for the determination of the hot and frozen spots in the design are is laid.

All these products of the domain analysis aere completed by records in form of class descriptions as well as the listing and documentation of their methods, attributes and relationships.

3.3 Design of the Simulation Framework

With the class diagrams defining the problem space as the main artifacts developed during the domain analysis, the design of the framework for discrete event simulation is refined iteratively. All important aspects of the problem domain are covered by documenting the solution classes needed together with their mutual relationships.

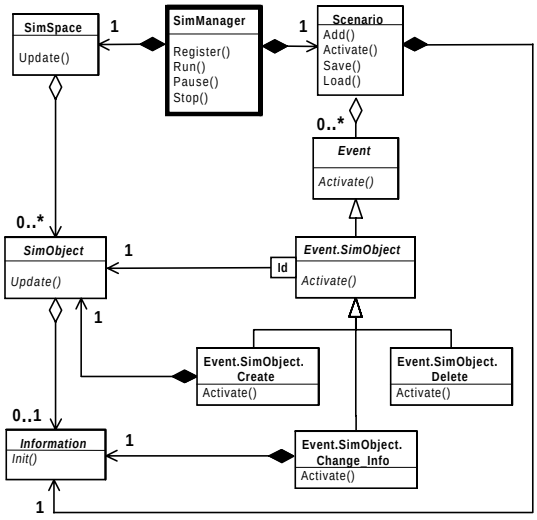


Figure 3: Design of the Framework

It became obvious during domain analysis and in early design of the simulation framework, that it is essential for its success to separate the parts which change often and therefore have to be adapted (i.e. the hot-spots) from the parts which are more stable (the frozen spots) and therefore need no adaptation. This process is based mainly on the domain experts knowledge of the aspects which differ from application to application and is called hot-spot analysis. The flexibility of the behavior desired as well as the time when it is needed have to be taken into account at this stage. Pree describes the hot-spot driven approach in [Pree 1994] with more detail.

The products of this phase are the framework design model, mainly as class and sequence diagrams in the Unified Modeling Language, with the associated documentation.

Figure 3 illustrates the design of the framework in form of a class diagram revealing the important classes, their essential methods, relationships, and cardinalities.

3.4 Framework Implementation

The implementation of the framework is tightly coupled to the programming language selected for the realization. In the case of the framework for discrete event simulation Ada95 was selected due to reasons beyond discussion in this paper.

In a first step the concepts of object-orientation were mapped to idiomatic constructs in the programming language selected. This task and its results are described elsewhere in more detail [Schmitt 2000]. The classes elaborated in the design phase together with their attributes and methods were translated to packages, record components, and procedures or functions. By convention, exactly one class was mapped to one Ada95 package. Resulting from this activity are the deployment diagrams, the source code of the framework, and the appropriate framework usage documentation.

3.5 Instantiation of the Framework

When a programmer wants to construct a specific simulation application on the basis of the proposed framework and the other components of the product line, he or she firstly has to instantiate singular objects from the concrete classes `SimManager`, `SimSpace`, and `Scenario` from within the simulation control component. It is also responsible for providing any interactions with an user via the display and property control components as well as the system under development or test via the interface component. The singular object simulation manager (instantiated from class `SimManager`) controls exactly one scenario and also has only one simulation space (objects instantiated from classes `Scenario` or `SimSpace`). The scenario object is linked to an object derived from class Information which comprises all necessary information on the scenario.

One or more concrete classes for simulation objects have to be derived from the abstract class `SimObject` by the application programmer using the framework in order to build a complete application. These concrete classes are then used to instantiate simulation objects as needed either during compile or run time. The abstract class `Event.SimObject` is in a similar way the basis for one or more concrete classes which describe the events needed and provide the templates for event objects used in the simulation. For each event object an information object has to be instantiated from the concrete class `Information` and linked to it.

4 Lessons Learned

Drawing from the experience with the domain analysis, design, and construction process of the object-oriented framework for discrete event simulation in the context of a product-line, the following findings are noteworthy:

- The implementation of the relevant entities and their relationships in the domain under consideration as a framework enabled the efficient reuse in a number of different but similar contexts.
- Having concentrated on the essential properties of the domain, a relatively small framework was build. This lead to a flattened learning curve for the application programmer and unproblematic use and maintenance of the component. An application programmer can easily grasp the basic concepts of the domain and use them efficiently without being a domain expert or reinventing the wheel.
- It needed several revisions of the frameworks design and implementation to get to a solid and viable solution.
- The flexibility gained through the use of an object-oriented framework in different solutions is very high. Many different applications can be build with this and the other components of the product-line.
- Documentation is a crucial point in the reuse of a framework. Without adequate description of the classes to use and the mechanisms to employ, it is very difficult to apply the component efficiently.
- The Unified Modeling Language can be used for describing the process, analysing and defining the problem domain, as well as detailing the design. There is no need for a special process description language or notation.
- The framework fitted into the environment of other product-line components even so these components were constructed with different programming languages and technologies.

5 Summary

This paper summarized the process and products of an industrial effort to construct an object-oriented framework for discrete event simulation in the context of a larger software product-line for a large spectrum of different simulators.

After giving an overview of the circumstances, goals, and the environment of the project, the process developed and used to design and construct a framework was illustrated. Starting with a domain analysis, the resulting model and documentation were transformed into a design model. This was used as the basis for the framework instantiation to build the final application component.

Finally, the lessons learned during this project were compiled and explained.

6 References

[Beck 1989] Beck, K, Cunningham, W.: A Laboratory For Teaching Object-Oriented Thinking, Proceedings of Conference on Object-Oriented Languages and Systems OOPSLA'89 (1989)

[Booch 1994] Booch, G.: Object-Oriented Analysis and Design with Applications, Addison-Wesley, Redwod City (1994)

[Booch 1999] Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide, Addison-Wesley, Reading (1999)

[Bosch 2000] Bosch, J.: Design and Use of Software Architectures, Addison-Wesley, Harlow (2000)

[Coplien 1999] Coplien, J., Hoffman, D., Weiss, D.: Commonality and Variability in Software Engineering, IEEE Software, Vol. 15, No. 6, (November/December 1998)

[Fayad 1999] Fayad, M., Johnson, R.: Domain-Specific Application Frameworks, Wiley, New York (1999)

[Jacobson 1999] Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process, Addison-Wesley, Reading (1999)

[Keller 2000] Keller, H.B., Plödereder, E. (eds.): Reliable Software Technologies Ada-Europe 2000, Lecture Notes in Computer Science 1845, Springer, Berlin, Heidelberg, New York

[Kruchten 2000] Kruchten, P.: The Rational Unified Process - An Introduction, 2nd edition, Addison-Wesley, Reading (2000)

[OMG 1999] Object Management Group: UML Specification, Version 1.3, (1999)

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

[Pree 1994] Pree, W.: Design Patterns for Object-Oriented Software Development, Addison-Wesley, Wokingham (1994)

[Prieto-Diaz 1991] Prieto-Diaz, R., Arango, G.: Domain Analysis and Software System Modeling, IEEE Computer Society Press, Los Alamitos (1991)

[Rumbaugh 1999] Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual, Addison-Wesley, Reading (1999)

[Schmitt 2000] Schmitt, D.: Analysis, Design and Implementation of an Object-Oriented Frameweork in Ada 95 in [Keller 2000]

[Weiss 1999] Weiss, David M., Lai, Chi Tau Robert: Software Product-line Enginnering, Addison-Wesley, Reading (1999)

[Wirfs-Brock 1990] Wirfs-Brock, R., Wilkerson, B., Wiener, L.: Designing Object-Oriented Software, Prentice-Hall, Englewood Cliffs (1990)