Developing Frameworks for Protocol Implementation

Ciro de Barros Barbosa[1], Luís Ferreira Pires

Centre for Telematics and Information Technology
University of Twente P.O. Box 217, 7500 AE, Enschede, the Netherlands
e-mail:{barbosa,pires}@cs.utwente.nl

Abstract. This paper presents a method to develop frameworks for protocol implementation. Frameworks are software structures developed for a specific application domain, which can be reused in the implementation of various different concrete systems in this domain. The use of frameworks support a protocol implementation process connected with formal design methods and produce an implementation code easy to extend and to reuse.

1 Introduction

The scope of our work is the investigation of the use of frameworks for protocol implementaion. Frameworks are software structures developed for a specific application domain, which can be re-used in the implementation of various different concrete systems in this domain. We are particularly interested in an implementation method that is connected to formal design techniques. In this case, the application domain of our frameworks is the design model underlying the design process; the concrete systems are the protocol implementations. A design model consists of the design concepts applied to model the protocol and the rules to combine such concepts [6].

The protocol implementation process consists of mapping the concepts of the chosen design model onto the concepts of an implementation model. An *implementation model* consists of the implementation building blocks available in a specific environment. In this work we use the term *meta-model* to generally refer to design models and implementation models. Ideally, for a precise mapping, both meta-models should have their semantics formally defined and a couple of tools should support the mapping process. However, implementation models do not have a formal semantics and the mapping process relies on the knowledge designers have of the implementation environments and their skills to combine the available building blocks. Implementations from scratch in these circumstances are not an efficient process. Implementation conformance usually relies on timing consuming tests without many guarantees.

The use a framework to implement a specific protocol, the final implementation will consist of the framework complemented with classes that define the specific functions of the protocol. In this respect a framework resembles a program with a ‘hole’, which has to be ‘filled in’ with specific functions in order to be executed [7].

By using frameworks we intend to give a contribution to the implementation process, providing an implementation method with a good balance between implementation process effectiveness, maintenability and performance. Such method also allows, to a large extend, the reuse of the implementation effort.

The framework development process must be committed with the following goals, in order to the resulting frameworks efficiently contribute to the protocol implementation process:

- frameworks must have a good balance between *code maintenance* and *performance*.

1.Sponsored by CNPq - Brazil

- frameworks must have a clear and intuitive *documentation*.
- there must be a *conformance* between frameworks and the concepts they implement.

We have defined a systematic approach to develop frameworks through which we intend to reach these goals. The analysis and design techniques used here were inspired in [2]. The modelling techniques follow the UML standard [8]. Our approach consists of four steps: *Domain analysis*, *Definition of the Software Architecture*, *Coding* and *Documentation*. The remaining of this paper describes in detail our method to develop frameworks.

2 Domain Analysis

In the domain analysis step we strive for a clear definition of the design model we intend to support. The concepts of the design model and their relationships are analysed and presented. Such analysis is based on a *basic meta-model* we have identified. The analysis consists of specializing the components of the basic meta-models into the specific characteristics of the particular design model we intend to use.

The modelling techniques we use in order to describe design concepts and their relations are *data dictionaries* and *class diagrams*. The data dictionary consists of one paragraph for each design concept, describing the semantics of this concept and its associations to other concepts. Such description is done in natural language. We use class diagrams to help describing the static relations between concepts. The next subsection presents our basic meta-model and discuss the specialization of its elements.

2.1 A Basic Meta-Model

In order to facilitate the analysis of design models, we provide in this sub-section a basic structure for meta-models. The basic meta-model, as we call it, consists of the fundamental concepts that can be found in any meta-model. The analysis of a specific meta-model can be done identifying how this specific meta-model defines each fundamental concept of the basic meta-model. We have obtained the basic meta-model by analysing the communities between the concepts present in some architectures for distributed systems, e.g., ISO/OSI [4] and Internet [5]. The same basic meta-model can be found by analysing description techniques which were conceived to model distributed systems [3]. The concepts identified in the basic meta-model are:

- *Behaviour*: is an abstraction which models the functionality of the system, i.e., how the system reacts, in each stage of its execution, to stimulus from its environment or the absence of them.
- *Component*: communication system architectures are often structured in layers. These layers may be decomposed into different functional parts. In this way we can reason about the system through smaller parts that are easier to understand than the whole. We use the concept of component to model such a system part. A component encapsulates the characteristics of the system part it models, e.g., its behaviour.
- *Interaction*: this concept models an instance of common activity between system parts.
- *Interaction means:* is an abstraction that represents the mechanisms through which components cooperate. The relation between interaction means and interaction is analogous to the relation between component and behaviour, i.e., it supplies a place and an identity for the interaction being performed.

For each specific meta-model we specialize the concepts presented above or aggregate new concepts to support them. Some of the basic meta-model concepts may have a few most common specializations, while others may rely on definitions that can vary considerably from meta-model to meta-model.

3 Software Architecture Definition

A modelling notation has to be used to document the resulting model, i.e., the framework software architecture. For the purpose of defining the software architecture, the modelling notation can still abstract from implementation details like language syntax and communication sub-systems interface. However, the mapping of the constructs of the modelling notation onto implementation constructs of an implementation environment should be straightforward.

The deliverable of this phase consists mainly of *data dictionaries* and *class diagrams*. *Interaction diagrams* [8] are eventually used to represent scenarios that are relevant for the understanding of dynamic aspects of the software architecture. In order to deal with more complex dynamic behaviour, i.e., an object class with a non-trivial number of states, we make use of *state diagrams* that offer a complete and precise behaviour description.

For each class we must also define its boundary conditions, i.e.; initialization, termination and failure. In the initialization, the system must be brought from a quiencent initial state to a sustainable steady state condition. Termination usually consists of releasing allocated resources. Failure can arise from user errors or from resources exhaustion. The benefits and drawbacks of the design decisions taken in the development of the software architecture are documented in natural language, in order to facilitate the selection among alternative frameworks, when implementing a specific protocol. The modelling process that defines the software architecture is mainly a creative process. Nevertheless, some heuristics can be applied to support this task. We present such heuristics in next sub-section.

3.1 Heuristics for defining the software architecture

By heuristics we mean any actions or procedures whose effectiveness has not been proven but are accepted based on experience or common sense. In software engineering, such approach still plays an important role and therefore saving such knowledge is a pragmatic attitude.

The main heuristic for defining software architecture comes from the object-oriented system development method presented in [2]. In order to reach the framework software architecture, the class diagram obtained from the domain analysis must be adapted to an implementation environment. Object classes are excluded or included in the software architecture for implementation optimization. New responsibilities of object classes must be defined. However, optimization of the design should not be excessive.

Another way to apply heuristics is the use of design patterns. A design pattern is a description of communicating objects that are customized to solve a general design problem in a particular context [1]. Design patterns are usually the result of a largely applied implementation solution which became well known and flexible for reuse purposes. The use of design patterns makes the software architecture definition more systematic. Design patterns represent a compromise between an one-class implementation solution, e.g., a linked list, which is reused the way it is, and a more complex solution which falls in the category of a framework.

For some important issues of protocol implementation, e.g., execution flow control and mapping of protocol behaviour, some design patterns can be found in the literature. For example, a pattern called scheduler can be used to share the opportunity of execution among different parts of a system; a pattern called event-demultiplexer to supply an event-driven execution model; the state pattern can be used for mapping a finite state machine.

4 Coding

In this step we map the framework software architecture onto the constructs of a specific implementation environment. Some instructions on mapping the object-oriented implementation model onto implementation code is given is the Object Design phase of the Object Modelling Technique (OMT) development method presented in [2].

We apply a prototyping strategy for coding. In this approach, the concepts which have stronger impact in the implementation are coded first. We start by producing a simplified prototype which allows preliminary tests. An interactive and incremental approach is also advised in the software development process presented in [9]. The prototyping strategy helps us reasoning about different problems separately. When dealing with some improvements in a framework, we can abstract from other features that are consolidated in previous versions of the same framework.

In the coding step we also have to decide about the trade-off between portability and performance. Building blocks that are intended to improve performance are usually less portable, for example, multi-thread packages. Some implementation building blocks offer portability, e.g., wrappers [10].

5 Documentation

Protocols are specified by creating instances of the concepts of a design model and combining these instances. Modelling techniques have constructs representing the concepts and constructs to combine them, as operators of concepts. Frameworks are intended to support more directly the semantics of design concepts, but it is not our goal to support any particular specification language. Therefore, in order to use a framework, we have to provide some guidelines that describe how to create instances of concepts and how to combine them.

Since we are applying object-orientation, instantiation of the constructs supplied by the framework to support design concepts can be done by instantiation of object classes. Some of theses object classes may need specialization in order to capture protocol specific information. In order to combine the instantiated concepts, different techniques may be applied, such as:

- template code to be filled in with code for each specific protocol function;
- suggested mappings of behaviour structures of the protocol specification onto implementation patterns.

The instructions for applying a framework must also contain the information for handling boundary conditions. Guidelines for using frameworks are formulated in natural language and illustrated with some examples.

6 Conformance Assessment

Protocol development methods that use formal methods emphasize precision as the basis for the design activity. An important requirement in our research is the conformance between design concepts and the way such concepts are supported by the protocol implementation. The semantics of the design concepts must be preserved in the implementation, otherwise, the effort done in the design may be compromised.

We have decided for a pragmatical approach to deal with the conformance requirement. We suggest testing techniques upon each concept implementation whenever the complexity of the concepts justifies so. We claim that the application of test techniques to basic design concepts is simple and produce reliable results.

7 Conclusions

We have briefly presented a systematic way to develop frameworks. This approach is an effort to combine formal design techniques with current practices in software engineering in a integrated protocol development process. We have presented the phases of our method and their deliverables. We have pointed out the conceptual basis of our work and the techniques applied. This is an approach that is better applied in environments where the changes in the required protocol functionalities are frequent. We claim that with this method we have addressed the cornerstones of an efficient and reliable process for implementing protocols.

8 References

[1] E. Gama, R. Helm, R. Jonson, J. Vlissides, *Design Patterns: Elements os Reusable-Oriented Software*, Addison-Wesley Publishing Company, 1994.

[2] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, *Object-Oriented Modeling and Design*, Prentice Hall, New York, 1991, 491p.

[3] *Using Formal Description Techniques: An Introduction to Estelle, LOTOS and SDL*, Edited by Kenneth J. Turner, John Willey & Sons, 1993, 431p.

[4] ISO, Information Processing Systems - Open Systems Interconection - Basic Reference Model, 1984, IS 7498.

[5] D.D. Clark, *The design philosophy of the DARPA Internet protocols*. In proceedings of the SIGCOMM'88 Symposium (Aug. 1988), pp. 106-114.

[6] C.A.Vissers et al., *The Architectural Design of Distributed Systems*, Lecture Notes, University of Twente, Enschede, The Nethedlands, 1995.

[7] C.B.Barbosa; L.F.Pires; M.Sinderen, *Frameworks for Protocol Implementation*, Simposio Brasileiro de Redes de Computadores 16, 1998,

[8] M.Fowler, K.Scott, *UML Distilled - Applying the Standard Object Modeling Language*, Addison-Wesley, 1997, 179p.

[9] G. Booch. *Object-oriented analysis and design with applications*. The Benjamin/Cummings Publishing Company, Inc., California, USA, 1994.

[10] D.C. Schmidt, *ACE: an Object-Oriented Framework for Developing Distributed Applications*, in Proceedings of the 6 USENIX C++ Technical Conference, (Cambridge, Masachusetts), USENIX Association, April 1994.