Integrating UML and OUN for Specification of Open Distributed Systems [*]

Wenhui Zhang
Institute for Energy Technology
N-1751 Halden, Norway

Einar B. Johnsen
Department of Informatics
University of Oslo, Norway

Olaf Owe
Department of Informatics
University of Oslo, Norway

Demissie B. Aredo
Institute for Energy Technology
N-1751 Halden, Norway

Abstract

Convenience in specification and possibility for formal analysis are important aspects in the specification of software systems. An approach that combines UML and OUN is proposed for such a purpose. The approach is demonstrated by a case study of an open distributed system.

1 Introduction

For development of open distributed systems, we need methods and tools for specification, design and code generation. For the specification, it is desirable to use graphical notations, so that specifications can easily be understood. On the other hand, it is desirable for the specification method to have a formal basis, in order to support rigourous reasoning about specifications and designs.

As no single existing methods cover all the desired aspects, we choose to extend, adapt and combine existing methods and tools into a platform for specification, design and refinement of open distributed systems, i.e. we try to integrate the UML (Unified Modelling Language) [3] modelling constructs, the OUN (Oslo University Notation) specification language [4] and the PVS (Prototype Verification System) specification language [5]. The purpose of the integration is to exploit the advantages of UML for high level specification with graphical notations, the textual notation of OUN for specification of additional aspects, and the theorem-proving capabilities of PVS for verification of correctness requirements.

This approach is demonstrated by a case study of a specification of the **SoftwareBus** system, an object-oriented data exchange system developed at the OECD Halden Reactor Project [1].

[*]This work is financed by the Research Council of Norway under the research programme for Distributed IT-Systems.

2 Specification of the System

The main motivation for constructing distributed systems considered here is the need that arises from the process industry. For surveillance and control of processes, data collected from processes have to be processed and presented. As the same set of data may be a basis for presentation in different forms at different locations, data sharing among different user-applications is a necessity. The architecture of such a distributed system consists of two parts: a *SoftwareBus* component and an unknown number of potential user-applications. The user-applications communicate with the *SoftwareBus* to carry out necessary data processing tasks including: creating and destroying variables, objects, and classes; assigning values to variables; and accessing values of variables.

UML Specification: The UML specification includes a description of the basic elements of the system like classes, components, and interfaces they provide to each other. Static structure and dynamic behaviour of the system are specified by putting together these basic elements into UML diagrams. We distinguish between the external interface and internal interfaces. The external interface specifies operations the *SoftwareBus* component provides as a service to user-applications. The operations of the external interface of the *SoftwareBus* component can be grouped into two parts: those concerned with data-object manipulations, and those dealing with connections between user-application components and the *SoftwareBus* component. Accordingly, we decompose the external interface into two sub-interfaces: *SB_SoftwareBusData* and *SB_SoftwareBusConnections*. These external interfaces and relationships between the *SoftwareBus* component and these interfaces are specified using the UML interface specification notations. Internal interfaces specify operations the different parts of the *SoftwareBus* component provide as a service to each other (not to user-applications). The *SoftwareBus* component consists of a central unit (hereafter called the **portmapper**) and a set of data servers. The purpose of the portmapper is to maintain information on the data servers, while the purpose of the servers is to store data that may be shared among user-applications. Within such a system, a user-application communicates with a data server for carrying out necessary data processing tasks. Depending on requests from the user-application, the data server may communicate directly with another data server for fulfilling the requests or communicate with the central unit if the information on other servers is requested or needed. Hence there are two interfaces: *SB_Portmapper* and *SB_DataServer*. The first one is the interface provided by the portmapper to data servers and the other is the interface provided by data servers to other data servers. The interfaces are similar to respectively *SB_SoftwareBusConnections* and *SB_SoftwareBusData*. The interfaces, the classes implementing these interfaces and relationships among the interfaces, the classes and the *SoftwareBus* component are specified using UML class diagrams and component diagrams.

OUN Specification: In contrast to OCL, interface declarations in OUN are used to specify relevant aspects of the observable behaviour of objects. An interface identifies a set of communication events reflecting the method calls for the currently considered role of the object. If a method "m" is implemented by some object o and it is called by another object o' with parameters $p_1, \ldots, p_n$ and and returns with values $v_1, \ldots, v_m$, the call is represented by two communication events: $o' \rightarrow o.m(p_1, \ldots, p_n)$ reflecting the initiation of the method call and $o' \leftarrow o.m(p_1, \ldots, p_n; v_1, \ldots, v_m)$ reflecting the completion of the call. The behaviour of the aspect of the object modelled by an interface is given by (first-order) predicates on finite sequences of such communication events. For each interface there are two (optional) predicates, an assumption and an invariant. The assumption states conditions on the environment of the object, i.e. a predicate that should hold for sequences ending with input to the object. The invariant guarantees a certain behaviour when the assumption holds, i.e. a predicate on the sequences ending with output from the object. The operations of interfaces in the UML specification are inherited in the OUN specification. In addition, assumptions on the environment of an object implementing the interfaces and invariants specifying the correct behaviour of the object are added. For instance, the requirement that a call by an object x to the method (i.e. interface operation) **sb_initialize** of the *SoftwareBus* y must either be the first event (in the communication sequence between x and y) or an event following the ending event of a call to **sb_exit** can be formalised as a predicate $A_1(x, y, h)$ on the communication history h:

$$A_1(x, y, h \vdash (x \rightarrow y.\mathrm{sb\_initialize}(\_))) == \\ (h/\{x, y\} = \epsilon) \vee (h/\{x, y\} \ \mathbf{ew} \ x \leftarrow y.\mathrm{sb\_exit}())$$

where the symbols "ϵ", "$\vdash$", "**ew**" and "_" denote respectively the empty trace, the *right append* operation, the *ends with* predicate, and an uninteresting parameter. The meaning of h/s is the projection of h on the events involving objects of the set s. This predicate is added to the interface *SB_SoftwareBusData* as an assumption.

3 Concluding Remarks

The advantage of using UML notations is that these notations are intuitive and are commonly accepted and used in industrial software development. The use of UML is important for the description of the initial software requirements which is normally a result of discussions between users and systems analysts (or software engineers). The advantage of OUN is that it can express additional aspects not easily expressible in UML and OCL. In addition, OUN is a formal language and the OUN specification can be used in the formal analysis of the consistency of the specification. For further system development, the UML and OUN specifications can be translated into PVS specifications for verification [2] and the software can be developed in an environment [6] which integrates Rational Rose (a tool supporting UML from Rational Software Corporation) and the PVS toolkit in order to cover the software development process from system specification, design, verification, to code generation.

References

[1] T. Akerbæk and M. Louka. The Software Bus - an Object-Oriented Data Exchange System. Report: HWR-446, Institute for Energy Technology, Norway. 1996.

[2] D. B. Aredo, K. Stølen and I. Traore. An Outline of PVS Semantics for UML Class Diagrams. Proceedings of NWPT'99, Uppsala, Sweden. October 1999.

[3] G. Brooch, J. Rumbaugh and I. Jacobson. The Unified Modeling Language User Guide. Addison Wesley Longman Inc., 1999.

[4] O. Owe and T. Ryl. On Combining Object-Orientation, Openness and Reliability. Proceedings of NIK'99, Trondheim, Norway. 1999.

[5] S. Owre, N. Shankar and J. M. Rushby. The PVS Specification Language. Computer Science Lab., SRI International. April 1993.

[6] I. Traore, D. B. Aredo and K. Stølen. Tracking Inconsistencies in an Integrated Platform. Research Report 274. Department of Informatics, University of Oslo, Norway. 1999.