Formal Development of Open Distributed Systems: towards an Integrated Framework

Issa Traoré, Demissie Aredo, and Ketil Stølen

[1] Institute for Informatics, University of Oslo
P. O. Box 1080 Blindern, N-0316 Oslo, Norway
[2] Institute for Energy Technology
P. O. Box 173, N-1751 Halden, Norway
{ issat, demissie, ketils}@hrp.no

Abstract. This paper contributes to the discussion on the issues related to the formal development of open distributed systems. The deficiencies of traditional formal notations in this setting are highlighted. We argue that there is no single formalism exhibiting all the features required. As a solution, we propose a multiformalism platform that involves three formalisms: UML, OUN and PVS-SL. We discuss the motivation for the choice of these formalisms and the main research issues underlying this kind of platform.

Keywords: Formal Methods, Open Distributed Systems, UML, PVS, OUN, Multiformalism, Object-orientation.

1 Introduction and Problem Statement

Motivated by the need for modeling the dynamic features of object-oriented programming languages and openness in distributed applications, the study of open, dynamically extendable systems has become a very popular research area. In fact, since the late 80ies, much research within theoretical computer science has been directed towards this kind of systems. The emphasis has mainly been on semantic issues; in particular, how should such systems be represented faithfully and fully abstracted. This has, for example, lead to the development of the Pi-calculus [Milner *et al.*, 1992], and to new refinements of the Actor model [Agha *et al.*,1997]. Most of the early proposals have a strong operational flavour. More recent denotational approaches [Jagadeesan,1995], [Stark,1996] are rather technical, and in most cases directed towards the Pi-calculus.
The above mentioned research attempts to find mathematical models suitable to describe the semantics of systems. The emphasis in our work is not on the semantics of systems, but on formal system development. Existing formal development methods suffer from certain limitations which constrain their application to large scale projects, especially their esoterism is a serious obstacle. This fact is well expressed by Ralf Kneuper as follows: "Software development is done by people, not by machines. No matter how 'good' a development method is, it will

only be successful if the developers who are to use it are willing and able to do so" [Kneuper,1997]. Most specification techniques supporting the development of open distributed systems, such as the UML (Unified Modeling Language) [Rational,1997], lack the formal semantics and the various reasoning facilities underlying formal developement methods. Moreover, we are not aware of any conventional formal development methods that is able to fully handle the flexible, extendable and very dynamic features characterizing distributed systems of today. In RM-ODP [ISO-rmodp], formal description techniques such as LOTOS, Z, SDL and Estelle are proposed for the specification of the various viewpoints involved. But, as pointed out in [Dahl,1998], these languages are only partly satisfactory. For instance, we may use Z for the description of the static parts of the information viewpoint, but it is not suitable to deal with the dynamic aspects. SDL and Estelle give little support for formal reasoning. LOTOS is a flexible description technique, but in our opinion, mainly suitable for the design phase.
So, taking the previous remarks into account, the challenge is to build a platform exhibting various capabilities such as:

1. The ability to be grasped and used in an industrial context; this requires characteristics such as communicability and friendliness.
2. The ability to support the main aspects such as openness and dynamic reconfiguration exhibited by open distributed systems.
3. The ability to produce a formal specification which can be amenable to formal verification and validation.
4. The existence of efficient tool support, a prerequisite for a large-scale application.

We are not aware of any single specification technique or method which provides all these capabilities. One obvious solution is to build-up a completely new method from scratch. However, this is extremely costly. Instead, we propose a multiformalism approach where we adapt and combine already exisiting technologies. More explicitly, based on the evaluation of several existing methods and CASE-tools [Stølen *et al.*,1998], [Stølen,1998], we propose a platform based on UML and OUN (Oslo University Notation) [Owe,1999], for specification and refinement, and on the PVS-SL (Prototype Verification System-Specification Language) [Crow *et al.*,1995] for the semantic foundation.
In the following section we discuss the rational behind the choice of the specification formalisms underlying the platform. Then, we discuss some of the main research topics involved and finally we make some concluding remarks.

2 Choice of the Specification Formalisms Underlying the Platform

In this section, we give an overview of the involved formalisms and discuss the rational behind the choice.

2.1 UML

The choice of UML was dictated by the fact that it is built on an object-oriented framework and provides several capabilities such as extensibility mechanisms (e.g. *stereotypes*), *dynamic* and *multiple classification*, which are useful for the description of open distributed systems. In addition, UML provides an underlying methodology for specification and refinement, a graphical notation which contributes to communicability and friendliness, and very importantly, UML is an international standard for object-oriented modeling.

Support for open distribution: Being an object-oriented approach, UML provides several capabilities such as *encapsulation, data abstraction, extensibility, reusability* and *flexibility*, which are helpful in modeling open distributed systems.
Among the *extensibility* mechanisms, we can mention *stereotypes* for adding new building blocks, *tagged values* for creating new properties for existing constructs, and *constraints* for extending the semantics of a UML construct.
Concerning *data abstraction*, there is a complete separation between specification and implementation objects. This allows us to design in terms of interface and to enable the evolution of the system by replacing an object by an alternative implementation. An *interface* is a collection of operations which are used to specify service of a class or a component.
In most object-oriented languages, objects are statically typed, so their types are bound at their creation time. In UML, this is expressed by class diagrams. In addition, there are mechanisms for handling the dynamic nature of an object type, which can be helpful in modeling *dynamic reconfiguration* in the context of open distribution. This is achieved through the multiple interfaces that a class may implement. An instance of such a class will support all of those interfaces, but according to the context, it may present only one or more of them as relevant. Each of those interfaces represents a role that an object can play over time. For instance, figure 1 is extracted from the specification of a mobile telephone system consisting of one central telephone exchange (not represented in the figure), two switching stations S_1 and S_2, and mobile telephone T in a car moving around. Each station covers different area. The telephone should always be in contact with one of the stations, which is at that time the base station, the other station being idle. In figure 1, we define a class *Station* and its different roles by defining two interfaces: *Base* and *Idlebase*. In an association between classes *Station* and *Telephone*, *Station* plays the role s, whose type is *Base*; in another association *Station* may play another role, say as *IdleBase*. Dynamic typing can also be rendered through an interaction diagram, by displaying the role of each instance of the corresponding class in brackets below the object's name or by connecting each variant with a *become* message. For instance in figure 2 (extracted from a collaboration diagram describing the previous mobile phone system), object o, of type *Station*, changes its role from *Base* to *IdleBase*. During interaction, the change in an object attribute values, states, roles or relationships can also be

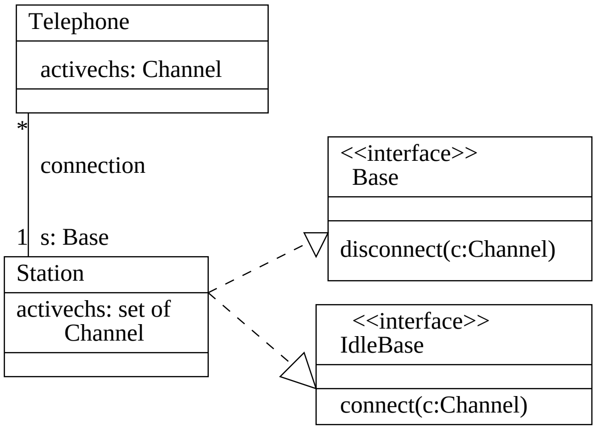

Fig. 1. Dynamic Typing through Class Diagram

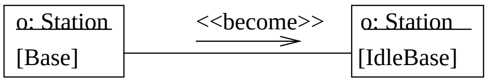

Fig. 2. Dynamic Typing through Interaction Diagram

modelled by attaching to it specific constraints, such as *new*, *destroyed* or *transient* to specify respectively creation, destruction and modification of the object. UML also provides several facilities for modeling distributed architecture, especially *component* and *deployment diagrams*. A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. A deployment diagram consists of nodes which represent the physical deployment of components; a node can be a processor or a device. We use nodes to model the topology of the hardware on which the system executes. We use component diagrams in conjunction with object diagrams and interaction diagrams (as mentioned previously) to model mobility. For instance figure 3

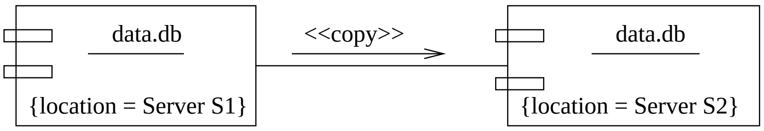

Fig. 3. Modeling Migrating Components

shows a system consisting of migrating components. For load balancing purposes and failure recovery, the system consists of databases replicated accross several nodes.

Limitations: In spite of the benefits provided by UML, there are several limitations in the context of the formal modeling of open distributed systems. The graphical constructs provided by UML are not enough to achieve a complete and precise specification of the system. For instance, in [Evans,1998] several incompletnesses in the static semantic model of UML are reported, especially concerning the definitions of the concepts of aggregation, inheritance, constraints on inheritance hierarchies and abstract operation descriptions. In order to fill this gap, there is a need for extending the capabilities of the UML with respect to two main objectives:

1. The description of additional constraints about the objects in the model, such as invariants on classes and types, abstract definitions of operations and attributes, non-functional requirements, etc.
2. The definition of a formal semantics for the different constructs involved, in order to remove all ambiguities.

The first of these objectives is generally performed using natural language resulting in ambiguities. An alternative which deals with both issues is OCL (Object Constraint Language) [Rational,1997], a semi-formal constraint language easy to read and write, which is used to specify well-formedness of the modeling abstractions provided by UML. An OCL specification consists of a set of expressions. An OCL expression has a type and has no side-effect. OCL has several constructs for modeling types, classes, interfaces and associations, but its expressiveness is relatively limited. For instance, operations with side-effect are not handled by OCL. Moreover, it is not possible to write program logic or control flow in OCL; it is not possible to invoke processes or activate non-query operations. In fact, as pointed out in [Evans,1998], the semantic of OCL is not mathematically defined, and hence it does not provide the facilities required for formal analysis: there is at most a set of type conformance rules.
Hence, instead of basing our platform on OCL, we have decided to use two other formalisms, OUN and PVS-SL, which are well-suited each for one of the two objectives mentioned earlier.

2.2 OUN

One of our objective in this platform is the production of abstract descriptions of systems. Trace-based notations are very efficient for this purpose [Jonsson,1987]. However, most of the existing trace-based notations don't support object-orientation, openness and dynamic reconfiguration; thus the choice of OUN for this platform. OUN is a formal development method which takes the deficiencies of traditional formal notations into account by addressing the main aspects of open distributed systems. Used in conjunction with UML, it can describe the invariants and constraints attached to the main constructs of UML such as types, classes, interfaces and associations. The main properties of objects such as attributes, association-ends, operations (with or without side-effect) can be expressed in OUN. In addition, the extensibility mechanisms of UML that serves to define new UML notions matches the specific needs of OUN. In contrary to OCL, OUN addresses the main implementation issues at abstract level. The main concepts considered in OUN include:

Objects with internal activity and structure.
Interfaces with syntactic specification of methods.
Classes with state variables and imperative style implementation.
Contracts used to restrict the interactions among a set of objects.

Inspired by Java and CORBA, OUN considers high level object-oriented concepts, and is oriented towards practical specification, rather than operational semantics [Dahl,1998]. Objects are specified by means of invariants on historic information: finite or infinite sequences of parameterized events that describe interactions between the object and its environment. Consequently, only information visible outside the object, such as its signature and operator invocations, is considered. Dynamic object creation and addition of interfaces, and multiple inheritance of interfaces and classes are supported.
An OUN requirement specification is provided in terms of interfaces and contracts. In contrary to UML, the concept of class appears later during design specification. An interface contains only the syntactic definitions of operations. It contains also a requirement specification taking the form of *assumption-guarantee* which may consist of an invariant asserting properties which each object that implements the interface should satisfy, and an assumption stating minimal context requirements. In contrast to UML, objects are typed by interface, this in addition to the possibility for an object to implement several interfaces, provides strong facilities for dynamic typing and hence for open distribution. We give, in the following, an OUN specification of a *contract* that specifies the interaction among objects of interfaces *Base*, *IdleBase* and *Telephone* defined previously for the mobile phone system.

contract Switch
begin
    with b: Base, ib: IdleBase, t : Telephone
    inv $\mathcal{H}/t$ **prs**$\langle[connect, disconnect]^*\rangle$
end

The invariant states that a request for a connection (*connect* message) should be followed by a *disconnect* message. $\mathcal{H}$ denotes the global communication history; the projection of the history onto an object o, denoted by $\mathcal{H}/o$, corresponds to the sequence of method-calls involving object o since its creation. Keyword **prs** is an abbreviation of "prefix of regular sequence".
A class contains definitions of attributes, implementation of operations and possibly an invariant and an assumption. An abstract implementation of the class *Station* is given below. Operators are defined using the guarded commands, in

which an unsatisfied guard represents waiting. The **with** clause states that only objects of the interface mentioned in the clause may interact with objects of the class through the listed operations. Keywords **ops, asm** and **inv** are used respectively for the operations, the assumption and the invariant defined in a class or an interface.

class Station
    implements Base, IdleBase
begin
    var activechs : **Set**(Channel)
    with T: telephone
        ops $connect(n : Channel) == \mathbf{true} \rightarrow activechs := add(activechs, n)$
            $disconnect(m : Channel) == \mathbf{true} \rightarrow activechs := sub(activechs, m)$
        asm ...
    inv ...
end

In OUN, it is possible to extend a class dynamically, by adding some operations and interfaces. This is another support provided by OUN for open distribution.

3 Integrating UML and OUN

3.1 Main Research Issues

The implementation of an integrated platform raises a number of research issues, among which, the following can be mentioned:

1. The identification of the interactions among the different formalisms involved, namely between UML and OUN, which give rise to a number of consistency proof rules. In [Traoré,1999b], we define a consistency relation that should hold between partial specifications developped using this platform.
2. The definition of the refinement proof rules.
3. The definition of the formal semantics assigned respectively to UML and OUN in PVS-SL.

Next, we discuss the last issue, namely the definition of the formal semantics of UML in PVS-SL; a discussion on the other issues can be found in [Traoré,1999a].

3.2 Formalising Object-oriented Models

Several works have attempted to provide a mathematical basis for the concepts underlying object-oriented models. Some of these approaches consist of adapting or extending a novel or existing formal description technique with object-oriented concepts [Moreira,1994]. Others derive a formal specification from the semi-formal (or informal) model built with exisiting object-oriented notations such as UML or OMT [Hayes,1991]. The main problem with these approaches is the fact that the user should have to deal with a certain amount of formal artifacts, and as we have already argued, this can be a barrier to an industrial use.

A third approach, that has been adopted in this platform, consists of assigning a formal semantics to an existing object-oriented notation [Evans,1998], [Evans,1998]. In this case, the formal "stuff" is hidden behind the graphical one. The user deals with the graphical model, while the formal "stuff" is processed automatically at the back-end.

In [Wing,1990], a formal language L is represented as a triple $(Syn_L, Sem_L, \mathcal{R})$, where Syn_L is a notation (the syntactic domain), Sem_L is a set of objects (the semantic domain), and $\mathcal{R}$ is a relation between them: $\mathcal{R} \subseteq Syn_L \times Sem_L$. $\mathcal{R}$ is based on precise rules that define which objects satisfy each specification.

Hence, since in our platform, we use the notation provided by UML and assign to it a formal semantic in PVS-SL, we define our satisfaction relation accordingly:

$$\mathcal{R} \subseteq Syn_{UML} \times Sem_{PVS}$$

For instance, in the case of UML class diagram components, the main semantic entities involved are PVS *theory* and *relation* concepts. Class and interface are both defined as theory that provides specific datatype definition.

An interface theory defines a record type whose fields are the signatures of its operations. A class theory defines a record type whose set of fields includes the declaration of the attributes and signatures of the operations. If the class is a subclass in some generalization relationships, then the record should include all the attributes and operations inherited. The record should also include the operations defined in the interfaces implemented by the class. Objects are defined as instances of the record type defined. A general scheme of a theory that represents a class is given below.

ClName : **THEORY**
[exportings]
BEGIN
  [type_declarations]
  IMPORTING [list_of_theorynames]
  *StructClName*1 : **TYPE** = [# list_of_local_attributes,
                list_of_signatures_of_local_operations #]
  StructClName : **TYPE** = **conc**(*StructClName*1, *StructImpName*)
  [functions_definitions]
  [some_invariants]
END *ClName*

The record *StructImpName* is a resultant record of concatenating all the records declared in the imported theories.

We define generically association as a parameterized theory labelled *GenAssociation*; we consider in this paper only binary association. An instance of association is defined as a relation (a set of ordered pairs) on objects of the classes involved. We give the following general scheme:

GenAssociation(C1, C2, R1, R2 : **TYPE**,
M1, M2 : **TYPE** = setof[nat]) : **THEORY**
BEGIN
Assoc : **TYPE** = setof[[obj1:C1, obj2:C2]]
AXIOM FORALL (obj1:C1):#{obj2:C2 | [obj1,obj2] $\in$ *Assoc*}$\in$ M2
AXIOM FORALL (obj2:C2):#{obj1:C1 | [obj1,obj2] $\in$ *Assoc*}$\in$ M1
END *GenAssociation*

The axioms constrain the number of objects of a class involved in an association that are linked to every objects of the other class. *C1, C2, R1* and *R2* are the datatypes corresponding to the classes and role names involved; *M1* and *M2* represent multiplicities. In order to put everything together, we define a class diagram as a theory which imports all the theories corresponding to its components and includes the different constraints defined upon them.

4 Concluding Remarks

One of the main objectives of our platform is to keep the formal "stuff" the user of the platform should have to deal with to the minimum possible. This in turn facilitates its industrial use. The OUN model which is provided as a complement to the UML model is concerned with specific aspects with reduced complexity, and hence easy to express. In this respect, we have decided to use PVS-SL in this platform, as semantics foundation and not as a specification language. As a result, the user will not need to have an in-depth knowledge of the PVS formal notation and proof system. PVS-SL offers a very general semantic foundation and a set of powerful tools. It is highly expressive and offers several mechanisms for formal analysis. For instance, it is possible to express and reason about infinite traces within PVS-SL and this is very important since OUN is trace-based. Compared to OCL, PVS-SL provides stronger support for all kinds of operations (with or without side-effects). For instance, in OCL, operations can be modeled by a recursive expression, but it is the modeler's responsibility to ensure the well-definedness of the recursion. In PVS-SL, however, this is handled by the built-in construct: the *MEASURE* clause.
Another criteria facilitating industrial use is the automation of the platform. We are, currently, developping a supporting environment, to which we refer as an *integrator*. The *integrator* integrates existing tool supports for the UML and the PVS-SL and at the same time provides the functionalities they do not offer, in order to cover the whole development cycle from requirements capture to final code production (see [Traoré,1999a]).

References

[Agha *et al.*,1997] G. Agha, I.A. Mason, S. Smith, C. Talcott, *A Foundation for Actor Computation*, Journal of Functional Programming, 7:1-71, 1997.

[Crow *et al.*,1995] J. Crow, S. Owre, J. Rushby, N. Shankar, M. Srivas, *A Tutorial Introduction to PVS*, WIFT'95, Boca Raton, Florida, April 1995.

[Dahl,1998] O.J. Dahl, O. Owe, *Formal Methods and the RM-ODP*, Research Report, University of Oslo, Norway, 1998.

[Evans,1998] A. Evans, *UML class diagrams - filling the semantic gap (draft)*, Technical Report, York University, 1998.

[Hayes,1991] F. Hayes, D. Coleman, *Coherent Models for Object-Oriented Analysis.* In OOPSLA conference proc., Phoenix, AZ, October 1991. Communications of the ACM.

[ISO-rmodp] ISO-IEC JTC1/SC21/WG7, The Reference Model of Open Distributed Processing, http://www-cs.open.ac.uk/ m_newton/odissey/RMODP.html.

[Jagadeesan,1995] L.J. Jagadeesan, R. Jagadeesan, *Causality and True Concurrency: a data-flow analysis of the pi-calculus*, Proc. AMAST'95, LNCS 936, P. 277-291, 1995.

[Jonsson,1987] B. Jonsson, *Compositional Verification of Distributed Systems*, PhD Thesis, Uppsala University, Sweden, 1987.

[Kneuper,1997] R. Kneuper, *Limits of Formal Methods*, Formal Aspects of Computing (1997) 9: 379-394.

[Milner *et al.*, 1992] R. Milner, J. Parrow, D. Walker, *A Calculus of Mobile Processes part I and II*, Information and Computation, 100:1-77, 1992.

[Moreira,1994] A. moreira, R. Clark, *Combining Object-oriented Analysis and Formal Description Techniques*, In ECOOP'94, volume 821 of LNCS, Bologna, Italy, July 1994. Springer-Verlag.

[Owe,1999] O. Owe, I. Ryl, *The Oslo University Notation*, Research Report, University of Oslo, Norway, April, 1999.

[Rational,1997] *UML Documentation*, Rational Software Corporation, Santa Clara, CA, 1997.

[Stark,1996] I. Stark, *A Fully Abstract Domain Model for the pi-calculus*, Proc. LICS'96, P. 36-42, IEEE computer Society Press, 1996.

[Stølen,1998] K. Stølen, *A Comparison of Eleven Specification Languages*, Technical report HWR-523, OECD Halden Reactor Project, Norway, March. 1998.

[Stølen *et al.*,1998] K. Stølen, T.W. Karlsen, P. Mohn, H. Sandmark, *Using CASE-Tools based on Formal Methods in Real-Life System development of Distributed Systems*, Technical report HWR-522, OECD Halden Reactor Project, Halden, Norway, March, 1998.

[Traoré,1998] I. Traore, *Application Characterization and Multiformalism in Software engineering*, PhD Thesis, LAAS-CNRS 7 Avenue du Colonel Roche 31077 Toulouse, France, May 1998.

[Traoré,1999a] I. Traoré, K. Stølen,*Towards the Definition of a Platform supporting the Formal Development of Open Distributed Systems*, Research Report, University of Oslo, Norway, January, 1999.

[Traoré,1999b] I. Traoré, D. Aredo, K. stølen, *Tracking Inconsistencies in an Integrated Paltform*, Technical Report, University of Oslo, Norway, submited to ASE'99 conference, April, 1999.

[Wing,1990] J.M. Wing, *A specifier introduction to Formal Methods*, IEEE Computer,vol 23,no 9,Sept 90,P8-24.