Towards a “Good” Functional and Executable Behavior Model.

D. Sidou, Institut Eurécom, France (sidou@eurecom.fr)

The objective of this paper is to present the more important features of a behavior model intended for the specification and the validation of distributed applications or distributed application components. Distributed application components are the basic building blocks that are directly useful for the procurement of working distributed applications, e.g. telecommunication management network (TMN) applications, electronic commerce applications.... Such distributed application components are typically built on top of distributed object computing (DOC) systems that basically provide the communication infrastructure. Among the existing DOC systems we are particularly interested in the OSI Systems Management (OSI-SM) framework [13] typically used for TMN applications and the general purpose OMG-CORBA [12] DOC system. There is an important requirement for a better specification of distributed application components. Indeed, without a precise, unambiguous and correct specifications it is difficult to build a truly interworking distributed application. To this end, it is necessary to go far beyond the communication infrastructure for which a reasonable level of maturity is available. The challenge is to provide effective specification and validation frameworks for application oriented issues, e.g. what is the meaning or semantics of each interaction occurring between a client and a server. As a matter of fact, organizations such as the network management forum (NMF) and the OMG have proposed new concepts to allow the standardization of application oriented issues. In the NMF we have *Ensembles* [11], in the OMG we have *Object Frameworks* [17]. A NMF-Ensenble defines a coherent grouping of object interfaces and utilization scenarios corresponding directly to a given network management application. However an Ensemble does not include any behavior specification of involved managed objects. Object frameworks are a more recent proposal by the OMG intended to cover behavior issues. In this paper, the goal is to define a suitable behavior model based on executable specifications to allow the validation of distributed object frameworks.

In terms of expressiveness of the specification framework, nondeterminism and dynamism are considered as the two important features in the behavior of distributed applications. Section 1 gives a more precise description of these features and presents the way they are considered in the proposed behavior model. In terms of validation involvement of users is the key point. Section 2 presents some motivations for this statement and describes also the way usability is achieved for validation issues.

1 Specification Framework

1.1 Nondeterminism

Nondeterminism is a natural consequence of distribution. In a distributed system, actions[1] may execute concurrently (concurrency), actions may execute in an undetermined order (unordering), or actions can be selected for execution in a non-determined way (choice). As a result, the overall behavior of the system can not be uniquely determined a priori. Consequently, a behavior model for distributed applications has to allow the modeling of nondeterminism. In fact two approaches can be distinguished depending on the use of control abstractions to organize the specifications of actions in the system.

1.2 Transition Systems

In this section the notion of *transition* and *state* are defined. Putting this two notions together defines in its turn the concept of *transition system* which can be viewed as a low level representation for the behavior of any distributed system. Transitions in the system are merely its atomic actions, i.e. steps that can be observed in a single and coherent phase. The configurations of the system between its atomic transition steps define naturally its states. The overall behavior of the system can be defined either by all the sequences of transitions or states the system can go through from a given initial state s_0. This defines the concept of *transition system*. Though the behavior of a system can always be given as a transition system, such a representation is rather impractical because it is too low level. Most of the time transition systems are used by verification tools as a backend representation [2]. Because specification is a human activity the availability of higher level abstractions is mandatory to allow the specification of state and transitions – even if validation is typically performed at the low level representations based on transition system.

1.3 Using Control Abstractions

In many notations dedicated operators are introduced to model the causes of nondeterminism, e.g. concurrency or nondeterministic choice operators. These operators are applied to some rather sophisticated control abstractions such as processes as in process calculi [9], or automatas as in automata based specification languages [16]. Processes and automatas are intended to give a suitable partitioning of the control state of the system into well identified and manageable pieces. Actions are typically organized on such control structures, i.e. actions are specified based on the local (control) states that are defined in each process / automata in the system. Because of the emphasis that is put on control issues in the configuration of the system such models are referenced as control oriented models.

[1]An action is defined as in RM-ODP2 [15] very generally by anything happening in the system. One important point is that the granularity of actions is a design choice. An action need not be atomic, so actions may overlap in time.

1.4 Declarative Specification of Actions

In this approach, each action is declared one by one. Each action is self contained in the sense that it contains both a specification of the conditions required for its activation, a specification of its effects, and a specification of any other constraint that it may observe. It is important to note the key role played by the underlying data state of the system. Effects are typically specified as data state changes, and constraints are assertional conditions on data states that have to be verified at well identified places, e.g. pre- and post-conditions or general assertions. The enabling condition is based on a condition related to the data state. However it may also include a triggering event used to model interactions of the systems with its environment. For instance, a client that makes a request on a server object, or more basic things such as data state changes. Since the configuration of the system is defined only w.r.t. data abstractions, such models are referenced as data oriented models.

1.5 Synthesis

One of the first models proposed for the specification of concurrent systems was Dijkstra's guarded command language (GCL) [1], which is a typical example of a data oriented model. The semantics of such languages is based only on nondeterminism. For instance in Dijkstra GCL at each step one guarded command among the enabled guarded commands is nondeterministically selected to be fired. In general instead of atomic commands actions of any granularity are used, it follows that the execution model is typically based on the nondeterministic interleaving of enabled actions. In contrast, control oriented models can be viewed as refinements of data oriented models intended to model specific features of the real world. For instance, automata based languages are particularly adapted to the modeling of communication protocols. At each step, the local control state in each automata / process defines the set of enabled actions. The execution model then follows as before the same principle of nondeterministic interleaving of enabled actions. Detailed modeling of control is useful if related knowledge is available and relevant at a give stage of specification. This implies that the actual distribution of actions in the system is established as well as their relative sequencing within each process or automata. In ODP terms this implies that a significant part of the engineering viewpoint issues are fixed. A model of the actual distribution might be required if real time constraints are to be checked that typically depend on the parameters associated to the communications channels (e.g. network links) existing between the execution threads in the system. However, if one is only interested in the specification of functional issues, an engineering viewpoint model is absolutely not required. In fact this would overspecify the problem [7]. Therefore a behavior modeling framework, based on the declarative specification of actions, where control issues are specified minimally is more adapted for the purpose of functional modeling.

1.6 Dynamic Nature

Role Modeling In contrast to interfaces that are object centric, behaviors are of collective nature [8]. This is a natural consequence of modeling application oriented issues where the contexts of utilization of objects has to reflect the application requirements. Within an object framework basic objects are typically composed to form composite objects or object configurations. Each object is intended to fulfill a well identified role in the configuration. The role can be used as an identifier for objects [15] involved in specification of behavior, but this identifier has a high level semantics value w.r.t. the application. In other words, a role represent a view of an object for a particular purpose or context of utilization. So roles, by capturing utilization contexts provide a suitable modeling abstraction to take into account application requirements. In addition, role modeling allows dynamic subtyping, which is as shown below a very important feature of distributed applications.

Subtyping A role defines a subtype for an object in the sense that an object fulfilling a role is still compatible with the core specification of the object itself, i.e. it still observes the interfaces defined on the core object. However, the behavior resulting from the interactions on such interfaces with other objects is defined w.r.t. the role. For instance specific state information may be associated to a role and updated according to the purpose of this role. In addition, in the behavior of an object in a given role interactions with other objects are specified using object references based on the role identifiers available in the object configuration, e.g. an object in the *client* role interacts with an object in the *server* role. So behavior gains to be promoted at the role subtype level. The advantages are that (i) much more expressive and readable behavior specifications are obtained, and (ii) core object are kept as simple as they are, i.e. most of the time limited to their mandatory interfaces and attributes.

Dynamic Subtyping The dynamic subtyping character of role modeling results merely from the fact that roles may be associated dynamically to objects. ODP states clearly that in a composite object, the association of a component object with a role may result from the actualization of a parameter [15]. An object can in fact play several roles, and this set can change over time according to its evolution in the distributed application. An important point is that a usual object oriented mechanisms such as (static) inheritance is clearly not satisfactory to model the different variants of an object just because these variants need to be dynamically available. Note that the set of roles fulfilled may have to be kept consistent. For instance an object is usually not allowed to play the *reviewer* and *reviewed* role at the same time. Such inconsistencies are not caused by role modeling, they merely reflect properties of real life systems. Interestingly, role modeling provides very expressive means to specify such constraints. This has typically been used to model and analyze management policy conflicts [10].

1.7 Relationships

The concept of relationship follows directly from the need to model configuration of objects. However, this is only one way to do so, and in fact some authors prefer not to use relationships because they want to keep the emphasis on roles. A typical example is the *role model collaboration view* of the object oriented role analysis and modeling (OOram) software engineering method [14]. One problem with relationships is that they are often used for other purposes such as graph modeling and navigation on object graphs. Though relationships is an overloaded concept, in the context of TMN applications it turns out that the generic relationship model (GRM) [5] is suitable to define roles for the purpose of dynamic subtyping. Note that as in [4] only GRM templates related to information modeling viewpoint issues are used. The computational viewpoint issues in GRM are not usable as such. For instance relationship operation mappings can not be defined precisely using GRM. In addition they are defined only using CMIS services.

2 Validation Framework

Validation of a specification consists to check the correspondence between informal requirements and the formal specification. Three techniques are commonly used :

1. *inspection* consists merely for the people involved to cross-read their respective specifications.
2. *reasoning* consists to prove properties about a specification.
3. *execution* proceeds by executing series of tests and by observing their outcome.

Though inspection is a technique that can always be used because it is always possible, it suffers from severe limitations because it is a manual process directly limited by the capacities of humans. In contrast execution and reasoning provide some assistance for validation. However, in the end the quality of the validation depends only on the relevance of either the executed test cases or the properties that were proved. So in both cases all depends on the users involved in the validation process. Execution and reasoning have been opposed for ages. Executable specifications are often qualified as less abstract and less expressive that non-executable ones. In addition, it has also been argued [6] that, though executing individual test cases is useful, it is less powerful than proving more general properties. However, with declarative specifications it can be shown [3] that a comparable level of abstraction and expressiveness can be obtained. In addition, executable specifications are much more prone to the involvement of users. They allow a direct *touch-and-fell* approach about the features of a system. This provides an excellent communication vehicle between users, specifiers and developers. In contrast reasoning has not yet reached a satisfactory level of usability, because theorem provers still need intelligent indications from users to achieve complex proofs. Even then other important issues are the manageability, the presentation…of such proofs, which is still a difficult problem in the currently available tools. Note that to allow for the executability of declarative specifications of actions a precise execution semantics based on the nondeterministic interleaving of the execution of actions has been devised.

3 Conclusion

By deliberately limiting our ambitions to the specification and validation of functional behavior properties, we have come up to a behavior model particularly adapted to information and computational viewpoint modeling. The declarative specification of actions is intrinsically nondeterministic. Thus there is no problem to model nondeterminism, which is a basic feature of distributed applications. Another important feature of distributed applications is their dynamic nature. To this end it has been shown how roles provide a powerful information modeling abstraction to capture application oriented requirements. Roles define object subtypes dynamically available, object configurations and behavior labels, all that being directly linked to high level application issues. Finally, validation is based on the principle of executable specifications. This is a pragmatic approach that has been felt more usable than reasoning. The resulting behavior model is "good" because it allows to model the intended distributed application features, moreover it is reasonably usable for distributed application domain experts such as telecom engineers both in terms of specification and validation.

References

1. E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.
2. J-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. Cadp (cæsar/aldebaran development package): A protocol validation and verification toolbox. In *CAV*, 1996.
3. Norbert E. Fuchs. Specifications are (preferably) executable. Technical Report 92, University of Zurich (CS Dept.), 1992. Available at ftp://ftp.ifi.unizh.ch/pub/techreports/.
4. Management of the Transport Network – Application of the ODP Framework, ITU-T G851-01, 1996.
5. ISO/IEC JTC 1/SC 21, ITU X.725 : General Relationship Model.
6. I.J. Hayes and Jones C.B. Specifications are not (necessarily) executable. Technical Report 90-148, University of Manchester, 1990. Available at ftp://ftp.cs.man.ac.uk.
7. H. Jarvinen and R. Kurki-Suonio. DisCo Specification Language: Marriage of Action and Objects. In *Int. Conf. on Distributed Computing Systems*, 1991. Available at www.cs.tut.fi.
8. H. Kilov, H. Mogill, and I. Simmonds. *Invariants in the Trenches in Object Oriented Behavior Specifications*, pages 77–100. Kluwer, 1996.
9. LOTOS : A Formal Description Technique based on the Temporal Ordering of Observable Behaviour, ISO / IEC 8807, 1987.
10. Emil Lupu and Morris Sloman. Conflict Analysis for Management Policies. In *Integrated Network Management*, 1997. available at ftp://dse.doc.ic.ac.uk.
11. Ensembles: Concepts and Format, 1992. Network Mangement Forum.
12. Common Object Request Broker Architecture, 1996. Available at http://www.omg.org.
13. The OSI Systems Management Standards, ITU-T X.7xx Documents.
14. Trygve Reenskaug. *Working with Objects : The OOram Software Engineering Method*. Manning Publications, 1996.
15. Basic Reference Model of ODP – Part 2: Foundations, ISO 10746-2, ITU X.902.
16. C. C. I. T. T. Functional Specifications and Description Language (SDL). Rec. z.100-z.104, Geneva, 1984.
17. Bryan Wood. Draft green paper on "object model for services", 1997.

This article was processed using the LaTeX macro package with LLNCS style