3

Formal Support for ODP and Teleservices

Joubine Dutszadeh, Elie Najm

ENST- Networks Department
Ecole Nationale Supérieure des Télécommunications
46, Rue Barrault, 75013 - Paris France
E-mail: {joubine | najm}@res.enst.fr

Abstract

The provision of distributed services in an open, heterogeneous environment requires a robust methodology for the entire service life-cycle; from service specification to service operation. The Reference Model of Open Distributed Processing (RM-ODP) - an ISO/ITU standard - is an architectural model that lays the ground for rigorous approaches to service development. RM-ODP features a multiple viewpoint approach that significantly enhances the development of distributed services by separating concerns into five relevant viewpoints; enterprise, information, computation, engineering and technology. RM-ODP also uses and advocates the use of formal methods for the design of open distributed systems. We introduce ODP-RM and its viewpoints, give some relevant details of the information and computational viewpoints, discuss the formal support of ODP concepts and investigate the suitability of some specification languages such as OMT, LOTOS, Z, SDL and their extensions as candidate notations for ODP models.

Keywords

Open Distributed Processing, Service Specification, Information Model, Computational Model, Formal Methods.

F. A. Aagesen et al. (eds.), *Information Network and Data Communication*

1 INTRODUCTION

High speed networking and processing and the emergence of new types of information teleservices have impacted standards in system architectures. The OSI reference model of the ISO and ITU-T has played a pioneering role as a framework for system interconnection and has paved the way for more advanced interoperability. However, the motivations behind the development of OSI were limited in scope. Object-based architectures are now emerging with more ambitious aims. New dimensions are now considered in these architectures, like e.g., portability, reuse and genericity, manageability, ease of development, ease of integration, dynamic binding and reconfiguration, safe interactions, guarantee of QoS constraints etc. Object-based distributed computing is being established as the most pertinent basis for the support of teleservices on large, heterogeneous computing and telecommunication systems. Indeed, several important international organizations, such as ITU, ISO, OMG, OSF, TINA-C, are currently defining similar distributed object-based frameworks as a foundation for open distributed computing.

Open Distributed Processing (ODP) claims to be the most generic object-based architecture. It is defined as a common standardization effort to ITU-T and ISO. ODP is seen as a wide spectrum paradigm, suitable for a wide range of applications. Among these, one can mention: Distributed Systems (and, more generally, dynamically reconfigurable systems and applications), Teleservices, Network Management, Intelligent Networks, Computer Supported Co-operative Work, Office Automation, and Data Base Management Systems. The endeavour of the TINA-C consortium (Chapman et al. 1995) is perhaps the most important project for the construction of an ODP based framework for Telecommunication Information Architecture.

The ODP reference model (RM-ODP) is the foundational text for ODP. It defines a framework for service and application development in a heterogeneous environment. ODP prescribes that multiple descriptions, with different abstraction levels, should be provided for any given entity or component of a service under study and development. The ODP framework, thus, offers specific transparencies at the different abstraction levels. Five abstraction levels (or viewpoints in the ODP terminology) have been defined within ODP and are considered to encompass the different areas of concerns that need to be covered in the development process. Specifications have to be provided in each of these viewpoints using a corresponding viewpoint model or an adequate viewpoint language. The five ODP viewpoints are: enterprise, information, computational, engineering and technology.

Already in the early 80's, the ISO standardization committee for Open Systems Interconnection advocated the use of formal techniques to specify the protocols and

services of the OSI layered architecture. ODP has put an even stronger emphasis on the application of formal methods. Thus, it is interesting to discuss how the classical FDTs (LOTOS, SDL and Estelle) developed in the OSI realm have been used in this rather new context of ODP. It is also interesting to investigate how FDTs need to evolve to match the new requirements and the role that other techniques (e.g. Z, ...) can play. More generally, it is pertinent to analyse how formal and object oriented approaches can contribute to the formal support of the ODP reference model, concepts and viewpoints. The aim of the present paper is to discuss some of these issues. It does not intend to be a comprehensive treatment of the subject. It rather presents some ideas and raises some questions regarding these topics. It complements and updates some already published material on the subject (Stefani 1992, Bowman et al. 1995).

The paper is organized as follows: section 2 gives an overview of the reference model for ODP with an emphasis on the information and computational viewpoints. In section 3, we discuss the need of a formal support for ODP modeling concepts and present ongoing work on providing architectural semantics for RM-ODP. Then we discuss requirements for formally supporting ODP viewpoints and discuss the present status of standardized and other techniques in that respect. Finally, in section 4 , we make some concluding remarks.

2 OVERVIEW OF RM-ODP

2.1 The RM-ODP documents

The Reference Model for ODP (RM-ODP) consists of the following four parts (only the first part is not normative):

Part 1 - Overview provides a tutorial introduction and gives a guidance to the interpretation of the model.

Part 2 - Foundations defines the basic concepts and the analytical framework. It introduces the notion of viewpoint and defines concepts and vocabulary that are common to all ODP viewpoints (object, interface, action, composition, etc...). It also introduces the principles of conformance to ODP standards and describes the way they are applied.

Part 3 - Architecture is a prescriptive model. It defines the five ODP viewpoints and the ODP functions. It contains the conformance statement which characterizes *Open* Distributed Systems.

Part 4 - Architectural Semantics contains a formalization of the ODP concepts and viewpoints. It is divided in two subparts: Part 4 and Part 4 amendment. Part 4 proper gives a formalization of ODP basic concepts in terms of corresponding constructs of the different standardized FDTs. Part 4 amendment is concerned with the formalization of the different viewpoints and provides appropriate specification constructs in the different FDTs. Part 4 amendment also contains a direct formalization of the Computational viewpoint.

In addition to the four parts, the ODP group is developing a standard on the trader, a central component in ODP whose role is to match service requests with service offers. Other work items have also started recently, including the topics of interface references, type repository function and the Interface Definition Language (a joint work with OMG).

2.2 The ODP viewpoints

The **enterprise viewpoint** states the subject matter (a service, a component, an application, a system) considered for development (what is the thing being considered), defines its requirements at a strategic level (why it is being considered), identifies its stakeholders and other participating institutions and establishes their roles (who will be involved and in what role), and describes the legal environment and the set of rules and regulations that govern its exploitation (the rules that must be obeyed when the thing will operate).

The **information viewpoint** describes the information needed to represent the system under consideration. The Information model represents the first step of a translation process from the enterprise requirements to existing products and running systems. In an Information specification of e.g. a service, real-world entities need to be mapped, through an abstraction process, into elements of the information viewpoint called information objects. ODP is not yet prescriptive with respect to information modeling. Not only the model and the notation are not prescribed, but also ODP does not indicate the kind of information that is relevant to this viewpoint. For instance, in order to support system evolution, one can consider the description of the existing system, including its resources, its communication infrastructure, its present requirements and specifications to be highly pertinent. Thus, the information viewpoint can be very complex and large enough to encompass all other viewpoints. This may be the reason for the limited development of the Information model within ODP. The common understanding for a "minimal" information modeling is that it suffices to take into account the information entities directly supporting the system being considered. The information modeling process can be divided into three parts; (i) the specification of the information structure of the information objects, (ii) the specification of the

dependencies between the information objects and (iii) the specification of the operations and constraints that govern the possible dynamic evolution of the system considered as a collection of information objects. ODP introduces the concept of schema as a means to capture the state, the structure and the possible dynamic changes of information objects. Three types of schemas are defined by ODP; (i) *static schema* - which states the structure of the information objects, (ii) *invariant schema* - which provides time-independent constraints on the state of the objects, and (iii) *dynamic schema* - which describes the possible state changes of objects over time.

The **computational viewpoint** considers an abstract implementation of the system under consideration in terms of interacting objects, whereby location, access, distribution, and failure are transparent. The Computational model is the (abstract) language used to describe applications in the computational viewpoint: in the Computational model, an application is represented as a dynamic configuration of interacting objects. Objects are the key concept in the Computational model. A Computational object has a state that may be accessed externally only through interactions at its interfaces (we will hereafter, when there is no confusion, refer to computational objects simply as objects). An Object may possess interfaces (possibly many) which may be dynamically created and deleted. Interfaces are names of locations of interactions between objects. Objects may change dynamically their communicating partners by exchanging interface names. Objects may also create and delete other objects. There are two kinds of objects in the Computational model, namely, *basic* objects and *binding* objects.

Binding objects are devised to convey interactions between interfaces (of basic objects or other binding objects). In fact, in the Computational model, the programmer may choose one of the two ways for describing the interactions between interfaces: (i) either explicitly through a binding object, or (ii) implicitly without exhibiting a binding object. When specifying an explicit binding object, the programmer may incorporate the QoS requirements (order, timeliness, throughput, etc...) on the transport of the interactions supported by that binding object. Binding objects can be used to model sophisticated binding requirements between and a dynamically changing collection of interfaces. In contrast, in an implicit binding only two interfaces can be bound and no specific requirements are made on the transport of their interactions: interfaces interact by message passing with no explicit ordering or delay required on the transport of these messages. There are three kinds of interfaces for computational objects: *signal, operational* and *stream*. Signal interfaces are the most primitive: operational and stream interfaces can be modeled as special types of signal interfaces. A signal is an operation name and a vector of values (references to interfaces). A signal interface is an interface that emits and receives signals. An operational interface is an interface that can receive invocations and possibly react with result messages. Invocations and result messages are signals. An operational interface has a type which is roughly defined to be the type of the operations it can handle (where the type of an operation includes the types of its

return messages). A subtyping system allows for the safe substitution of an interface of a given type by another interface having a subtype of this type. A stream interface is an abstraction of a signal interface: the type of a stream interface is simply a name and a role (sender or receiver).

The **engineering viewpoint** determines how computational, distribution-free, descriptions can be realized in terms of generic system components and communication protocols. From an engineering viewpoint, an ODP system is considered as a collection of computer systems called nodes. At each node, computer resources are encapsulated in a nucleus (supported by the node), and the details of the underlying communication networks, operation systems and hardware are hidden by a uniform and basic distributed environment composed of inter-operating nucleis. This allows a realization of a node in multiple ways (from a technology viewpoint). Supporting the communication between computational interfaces, with specific Quality of Service constraints, is one of the most important functions provided in the engineering viewpoint. Different binding policies (connection based, connectionless, dynamic bindings with relocators, ...) may be used in the engineering model to realize the interactions between computational interfaces. Functions other than communication support (e.g., atomicity, dependability, ...) are also part of the engineering model. The identification of the relevant supporting engineering components and their standardization is a continuous process. New protocols will emerge satisfying new application needs (as identified in the computational viewpoint). Intelligent, ODP compliant (more or less automated), compilers will be devised and maintained to support the translation from the computational and information models of a system into its engineering representation. It is within the **technological viewpoint** that the generic engineering components are matched with existing pieces of hardware and software, thus providing real integrated products and implementations which comply with the specifications provided with the upper viewpoints.

3 FORMAL SUPPORT TO RM-ODP

Due to the inherent ambiguity of natural languages, RM-ODP modeling concepts may not be fully described by an informal natural language. While a formal specification language provides an unambiguous and precise notation, each user of a natural language specification could have a different interpretation of the specification semantics. Moreover, natural languages are known to become bulky very quickly and to not optimally achieve an iterative and incremental specification refinement. On the other hand, the constructs of a formal specification language can be mapped onto mathematical models or entities. Thus, it is possible to reason about certain properties of the specifications, and to prove that a consistency holds between separate specifications of a particular component made at different levels of abstraction.

Within RM-ODP, there are many areas where formal support reveals to be helpful. First, the foundational modeling concepts can be rigorously defined. Second, the viewpoint languages can be substantiated with abstract syntax and formal semantics. Alternatively, one can define viewpoint-oriented styles in existing specification techniques and assess their suitability as notations for viewpoint models. Third, the ODP functions (like, e.g. trading, type management) can be specified throughout the five viewpoints.

RM-ODP is Object-based and as such, it is impacted by research in the emerging research field of formal support for the OO paradigm. OO aims at encompassing those features that are believed to provide the adequate answer to the software crisis. For instance: modularity, compositionality, reuse, abstraction, encapsulation, separation of concerns, inheritance, specialization, delegation, typing, sub typing, classes, concurrency, dynamic reconfiguration, migration, etc... are recognized to be highly desirable in specification and programming languages. The need for these features has been established on pragmatical grounds and has been revealed by teams of developers. Until recently, however, the development of the OO approach has been driven mostly by software practitioners and without sufficient support of theoretical foundations. In particular, there has been no significant investigation on the compatibilities and/or conflicts that may arise when combining OO features in a single integrated framework. Many authors are now contributing to the establishment of mathematical based semantics for the different features of OO and of their integration (Nierstrasz 1993).

In the remainder of the section, we discuss the ongoing work within ISO on providing architectural semantics for RM-ODP, and then, we examine the issues of notations for ODP viewpoints, with special emphasis on the Information and Computational viewpoints.

3.1 Architectural semantics for RM-ODP

Formalising RM-ODP concepts

Part 4 of RM-ODP focuses on the formalization of the most fundamental ODP modeling concepts from Part 2. It mainly deals with two sets of concepts: (i) the *basic* concepts - including the notions of object, environment, action, interface, activity, behaviour, state, communication, location in space and time and interaction point; and (ii) the *specification* concepts - including the notions of composition/decomposition, behaviour compatibility, refinement, trace, class/subclass/superclass, type/subtype/supertype, template and interface signature. In Part 4, these concepts are equated with corresponding constructions in four FDTs, namely, LOTOS (process and data parts), Estelle, SDL and Z. The ODP modeling concepts are so general that their interpretation within the different FDTs is more a specialization exercise than a real formalization. Part 4 appears to be, in fact, a

multilingual dictionary for the ODP basic vocabulary. What it does not contain (and this is not the scope of the document) is a direct analysis of the relationships between the ODP concepts (it is up to the reader to deduce such information by studying a given FDT mapping). The issue of relating ODP concepts is partially addressed in (Gotzhein 1995) where the notions of interaction point, agent and refinement are studied. The main benefits gained from the Part 4 mapping exercise are a better understanding of the analysed concepts and the discovery of some inconsistencies that were used to improve the natural language text of Part 2.

Formalising RM-ODP viewpoints

Part 4 amendment deals with the formalization of the viewpoint languages. This work is still in progress. At present, Part 4 amendment contains a direct formalization of the Computational model and a mapping of the different viewpoint concepts on appropriate constructs from the four FDTs: LOTOS, SDL, Estelle and Z.

Like any interpretation exercise, the mapping of viewpoint concepts to FDTs cannot achieve the full formalization of these viewpoints. For a given mapping it is difficult to distinguish what describes the essence of the viewpoint model and what is FDT dependant. Moreover, the mapping should not be considered as a complete guide on how to use a given FDT as a viewpoint specification language. It only gives indications (and motivations) on possible associations between viewpoint language elements and corresponding FDT constructs. The validity of these associations remains to be assessed in complete workable examples. For instance, consider the proposed mapping of the concept of computational interface to that of a LOTOS gate (together with the behaviour visible at that gate). Knowing that the LOTOS parallel operator is static and defines a static communication structure between processes, how can one model the interactions between a client interface and a newly discovered server interface ?

The direct (FDT independent) formalization of an ODP viewpoint is highly desirable as it may serve as a pivot definition for comparing specifications written in different FDTs. Of the five ODP viewpoints, the computational viewpoint is the most defined in the standard. For this reason, its direct formalization is feasible. As of today, the formalization of the computational model with implicit binding is now stable (Najm and Stefani, 1995) and work is still in progress for the signal interfaces and explicit bindings (Najm and Stefani, January 1995).

3.2 Notations for the ODP viewpoints

The suitability of some notation as a viewpoint specification language has to be assessed by three sets of requirements: (i) the general requirements for specification languages; (ii) the ODP specific requirements; and (iii) the viewpoint specific requirements.

General requirements for specification languages

This type of requirements has been widely discussed in the literature. Among these requirements one can mention: simplicity of the semantical model (no superfluous complexity), executability and verifiability (useful tools can be constructed), expressiveness (what kind of requirements/behaviour can be captured: real time, probabilistic, true concurrency, etc...), modularity (easiness to combine pieces of specifications), openness to evolution (reusability and extendibility), user friendliness of the syntax (visual programming/specification), wide acceptance (standard de jure, de facto, ...), availability of tools (verification, documentation, test generation, code generation, ...).

ODP specific requirements for specification languages

Positioning in the ODP framework

Many specification techniques can be very powerful when considered by themselves but may fail in the ODP context because of their inappropriateness to support any viewpoint or because there has been no well identified requirement that they can fulfil. This is the case for FDTs built on strong architectural assumptions that do not quite match those of ODP (see, for instance, the discussion on the incompatibilities between reasoning and modeling approaches to software in the paper by P. Wegner in Agha et al. 1993). Therefore, the candidate notation should position itself clearly in the ODP framework. There should be a clear statement on its use in the ODP context and on the levels of abstraction that it claims to support. It should also provide user guidelines on how to approach specifications for a given viewpoint.

Articulation between viewpoints

The originality of ODP lies in its viewpoints and more importantly in the articulation between them. Indeed, the ODP framework is a wide spectrum scheme that encompasses the design and development trajectories advocated by many OO development methodologies. Thus, a candidate notation should not only serve its purpose as a linguistic support but it should also allow to relate specifications in other viewpoints.

The relations between specifications in the same or different notations have been studied in the recent years (Bowman et al.). Relations such as refinement, behavioural compatibility and implementation were defined and the general notion of consistency is also being investigated. These results may serve as a starting point for more advanced studies on how to relate the viewpoints themselves. Indeed, many issues are still open, like for instance, how to map the dynamic schemas of the information viewpoint to atomic operations of the computational viewpoint and from there to transactions in the engineering viewpoint. A challenging research topic is concerned with the translations of transparencies and QoS requirements

from the computational viewpoint into engineering mechanisms, involving communication protocols, admission control policies, resource management, and operating system scheduling. The concept of computational binding object is very important in that respect as it serves as a good articulation between requirements and generic implementations.

Object-orientation

Objects are used throughout the five ODP viewpoints, but different OO dimensions are emphasized in different viewpoints. Thus, a candidate notation should be object based and possess the OO dimensions appropriate for the considered viewpoint. Both Computational and Information modeling exhibit encapsulation and operation as essential characteristics of objects. But while information modeling aims at capturing the structure, the relationship and the global coherence of objects, computational modeling features object identity, interface reference, explicit interaction between objects (through message passing or binding objects) and dynamic reconfiguration. These issues are discussed in the sequel.

3.3 Information modeling requirements

There exist several notations and methods that are candidates for information modeling within ODP. An ideal notation for the ODP information model should support the principles of object-orientation and should also offer an easy-to-use interface to the service specifier for the task of mapping real world entities into information objects. Yet, this notation should have a formally defined semantics for the sake of analysability, tool support, and compatibility between the information specification and the specifications made in other viewpoints. Hereafter, we discuss three important requirements: ability to directly capture the information model, abstractness, and friendly graphical user interface.

Ability to capture the information model

There is an important debate concerning the scope of the information viewpoint. If one takes a minimalist position (which is what is currently being adopted by a majority of experts), an information specification should at least (and at most) identify the information needed to represent the objects of the system. In particular, it should capture: (i) the types and structures of information entities, (ii) the relationship between these information entities; and (iii) the allowed evolution schemes of the information entities which mirror the dynamic behaviour of the objects of the system. Ideally, a notation should be able to directly address the static, invariant and dynamic schemas defined in ODP.

Abstractness

A major requirement for an information modeling notation is to possess the appropriate level of abstraction. Information specifications should not be overspecified with details concerning, for instance, explicit interactions between

objects. Information specifications should also leave open all possible computational solutions, i.e., they should not bias towards any types of configurations of computational objects. Languages like Estelle and SDL, and to a lesser degree LOTOS, fail to be sufficiently abstract in that respect.

Graphical interface

Information structures and relationships can be naturally captured by a user friendly graphical notation. Graphical interfaces are also well suited to support incremental specification. As an information specification progresses, one can modify and refine objects and relations in an iterative fashion. These are the most basic graphic elements of the OMT (Rumbaugh et al. 1991) object model.

Discussion

Z is being considered as highly appropriate for information modeling. Although Z is not fully object oriented, it has those object features that match the needs of information modeling. Static, invariant and dynamic schemas can be directly mirrored in a Z specification. Moreover, Z provides a good basis for relating specifications in other viewpoints or languages (Derrick et al. 1996). Z has however some shortcomings: it is not graphical (or, perhaps, it is semi-graphical); its semantics, based on set theory, allows for limited tool support; and it provides no modularity to structure large specifications (see Johnson and Kilov, 1996 for a discussion on this last point). It should be noted, however, that there is a large community of Z users and researchers, and a lot of work is in progress to solve these issues (Object Z, Zest, and other extensions).

The object model of many OO analysis methods, like e.g. OMT or Fusion, are also well suited for ODP information modeling. Although they are not really formal, they have been used because of their appealing graphical syntax and because they are part of fully integrated development methodologies. It is worth noting that much research is now being carried out to endow these models with formal semantics, either directly (Dustzadeh and Najm, 1996) or through a translation to Z (B. Bates et al., 1996) or through a translation to an Abstract Data Types language (Bourdeau and Cheng, 1995). The three paradigms, Z, OMT and Fusion should be contrasted on the issue of modeling of actions. Fusion, like Z, and unlike OMT, allows to capture global actions (with pre and postconditions) describing the global behaviour of a collection of objects. Such a feature can be advantageously exploited in the refinement process from information to computation. Indeed, information actions can be refined, using a semantics preserving mapping, into a co-ordination of activities between computational objects. Global actions are also a prominent feature of the DISCO language and tools (Järvinen et al., 1990).

3.4 Computational modeling requirements

The computational model is a general framework that prescribes permissible interactions between (potentially) distributed objects and abstracts from their internal representation. The generality of this model is such that it encompasses a wide range of interaction paradigms, including RPC, client/server and continuous streams. Thus, there may be many contending notations to support this viewpoint; but none may really embrace it entirely. In fact, the specific requirements for the formal support of the computational viewpoint are diverse. In the sequel, we will discuss some of them: object basis, dynamic reconfiguration, synchrony and asynchrony, contracts, typing and subtyping of interfaces, and non functional behaviour.

Object basis

The computational model encompasses a full-fledged, object based abstract language. In Wegner's terminology, object basis means encapsulation of state and access to state (for read or update) only through operations on interfaces. This feature seems to be possessed by many languages (including Estelle, SDL and LOTOS) not known as being objet oriented. However, object basis is more meaningful when combined with dynamic reconfiguration. This, in fact, limits the number of possible candidates.

Dynamic reconfiguration

The computational model is highly dynamic. Objects may be introduced (and discovered by others), created and deleted. Their interfaces may be created and deleted. Their interacting partners may change over time. Dynamic reconfiguration is achieved by two mechanisms: exchange of interface identifiers between objects and creation of binding objects. The candidate notation should allow to naturally capture these two mechanisms.

Synchrony and asynchrony

A collection of computational object interfaces can interact when a binding, either *implicit* or *explicit*, has been established between them. In an implicit binding, the interactions take the form of messages exchanged through the ether. Thus, in this case, the communicating interfaces interact synchronously, i.e., the creation of the message is independent (in time and space) from its reception. In an explicit binding, a binding object is created between the bound interfaces. In that case, although the distant bound interfaces interact asynchronously, the local interaction between a bound interface and a binding object is synchronous. This special form of synchrony is a design choice for the computational model. Its aim is to favour the establishment of clear and rigorous contracts between basic and binding objects.

Contracts

The notion of contract is central to ODP. A notation should allow to relate between what is provided to an object from its environment and what , in turn, can be guaranteed by this object. A contract covers both the aspects of functional and non functional behaviour. The various dialects of temporal logic, enhanced with real time, seem well suited for this type of requirements.

Typing and subtyping

Interfaces of computational objects are strongly typed. An interface type delineates the set of operation names (with the types of their arguments and the names and arguments of their returns) available at the interface (or the type of signals in the case of a flow). Objects interact safely if their interfaces are related in a subtyping relation. Safe interaction means that their will be no errors of the kind 'operation not supported' at run time. The candidate notation, thus, should allow, to encompass a typing language. A higher order typing language may also prove useful for the description of some special purpose objects, like e.g., the trader which manipulates types as values. Compatibility with IDL is also highly desirable.

Non-functional behaviour

The real time and probabilistic dimensions are very important for the ODP computational model. Binding objects, for instance, describe the behaviour of communication which complies with certain Quality of Service constraints. These are expressible in terms of maximum delay and jitter, throughput, connection delay, real time synchronization, error and failure probabilities, etc. Non functional requirements are relevant, especially, when it comes to the support of multimedia interactions in distributed systems.

Discussion

The advent of ODP and its computational viewpoint is a new challenge for standardized FDTs. The main problem arises from the fact that Estelle, SDL and to a lesser extent LOTOS and Z, are based on strong architectural assumptions which are too specific and not fully compatible with those of the computational model. For instance, SDL and Estelle support FIFO communication and LOTOS limits the communication structures of processes by a static parallel operator.

The suitability of FDTs should be assessed not on a small set of specification exercises, limited in time, but on a real project that has to evolve with modified user requirements. Such evaluation has not really taken place yet. Some authors have found interesting approaches on how to handle computational specifications with LOTOS and SDL (Vogel, 1993 and 1994). These solutions represent the best that can be done with these two techniques rather than a real assessment.

Estelle and SDL'88 are both based on the Communicating Extended Finite State Machine Model. This model was very convenient in the realm of OSI (and more particularly for the lower layers). The possibility of typing interaction points was also

provided. However, these two languages fail on many other aspects as notations for the computational model. For instance, they do not allow to capture dynamic reconfiguration, their interface typing is not meaningful for interface trading, and they have very limited non functional features. SDL'92 is an object oriented enhancement of SDL'88. The main novelty of SDL'92 is inheritance: transitions of EFSMs can be constructed by inheriting other 'virtual' transitions. Thus, SDL'92, is not really a concurrent object-oriented language (see Prinz 1996 for ongoing extensions to SDL).

LOTOS inherits the object-oriented flavours brought by its ADT and process-calculus constituents. Its object-orientedness has been investigated by many authors. For instance, its ability to support (although not naturally) dynamic configuration has been shown in (Najm and Stefani 1991). Some form of simple inheritance has also been shown to be possible in LOTOS (Mayr 1988, Cusak et al. 1989) while more sophisticated forms can be accommodated using extensions (Rudkin 1992).

The situation with LOTOS should be considered in light of the ongoing work on its extension. In fact, since July 94, the Formal Description Techniques (FDTs) group within ISO has become part of the Open Distributed Processing (ODP) standardization committee (SC21/WG7). Thus, supporting the formal design of open distributed systems is a new objective for this group. The ISO-FDT team of experts is now working on the standardization of an extension of LOTOS - temporarily called E-LOTOS, which is targeted, among other things, at providing support to the design of ODP systems. As of now, dimensions like real time, modularity, executable data types and mobility are being considered for enhancements. MT-LOTOS, an integration of real time and mobility (based on Milner's et al. π-calculus) enhancements in LOTOS, has been used to compositionally specify a multimedia binding object (see Février et al. in the present proceedings). Work on E-LOTOS seems promising. It should be noted, however, that E-LOTOS is still a low level language and the notions of interface, interface type and operation are not primitive.

A considerable amount of research is being carried out relating process algebras with the object paradigm (Pierce and Turner, 1995). Actors (Agha 1986) and the π-calculus (Milner et al., 1992) have laid the foundations for this endeavour. It is not easy to predict how much of this work can be accommodated in E-LOTOS. An important last issue now being investigated by many researchers is concerned with high level abstractions for object co-ordination (Agha et al. 1993). Results in this area will impact both the computational and engineering models. Using such co-ordination languages, it is hoped that one can more easily handle mechanisms like atomic transactions or resource sharing in distributed systems.

4 CONCLUSION

In this paper, we have presented the ODP viewpoints and discussed the application of formal description techniques to the specification of services in the framework of

open distributed systems. The parallel effort between the definition of RM-ODP architectural concepts and the application of FDTs on these concepts has been a fruitful handshake for both RM-ODP and standardized FDTs. On one hand, formal interpretation of RM-ODP concepts has brought a considerable feedback to the correctness of RM-ODP modeling concepts, and on the other hand, due to the very broad view of RM-ODP architecture, FDT experts became aware of some shortcomings of the current standardized FDTs and new extensions have been envisaged and achieved. However, there are still many obstacles to overcome for a full support of RM-ODP concepts by formal description techniques. ODP is founded on concepts and paradigms that are now actively researched and developed. Formal support to ODP will certainly benefit from this activity. Existing standardized FDTs and their extensions will play a role. Due to the abstractness of the viewpoints and to their independence of any language, other techniques are also likely to emerge.

Acknowledgements

The authors would like to thank Jean-Bernard Stefani from CNET and the members of the ODP groups on Architectural Semantics and Extended LOTOS for the valuable discussions on topics related to the subject of the present paper. The authors would also like to thank SITA (Société Internationale des Télécommunications Aéronautiques) for sponsoring this work, and most particularly Gregory Ouillon from the R&D department for his valuable support.

References

Agha G. "Actors: A model of Concurrent Computation in Distributed Systems". The MIT press, Cambridge, Massachusets. 1986.

Agha G., Wegner P. and Yonezawa A. "Research directions in Concurrent Object-Oriented Programming". The MIT press, Cambridge, Massachusets. 1993.

Agha G., Frølund S., Kim W., Panwar R., Patterson A. and Sturman D. "Abstraction and Modularity Mechanisms for Concurrent Computing" in Agha (1993).

Bates B. W., Bruel J-M., France. and Larrondo-Petrie M. M. (1996) "Formalizing Fusion Object Oriented Analysis Models" – in FMOODS'96.

Booch G. and Rumbaugh J. (1995) "Unified Method for Object-Oriented Development" - Documentation Set, Version 0.8, Rational Software Corporation, 1995.

Cusak E. et al. "An Object-Oriented Interpretation of LOTOS" In Proceedings of Forte: Formal Description Techniques, December 1989.

Bourdeau R. H. and Cheng B. H. C. (1995) "A Formal Semantics for Object Model Diagrams" – IEEE Transactions on Software Engineering, October 1995.

Bowman H., Derrick, J. Linington P. and Steen M. (1995) "FDTs for ODP" - Computer Standards and Interfaces. Special Issue: Formal Description Techniques. Volume 17. N 5-6, September 1995.

Derrick, J., Boiten E., Bowman H., and Steen M. (1995) "Translating LOTOS to Z" - in (FMOODS'96)

Chapman M., Berndt H. and Gatti N. (1995) "Software architecture for the future information market" – Computer Communications, Vol. 18 - Num. 11, November 1995. Elsevier Science B. V.

Coleman D., Arnold P., Bodoff S. Dollin C., Gilchrist H. Hayes F. and Jeremaes P. "Object Oriented Development: The Fusion Method" - Prentice Hall, 1994.

Dustzadeh J., Najm E. (1996) "Direct Formalization of OMT Object Model" – ENST internal report.

Février A., Najm E. Leduc G. and Leonard L. (1996) "Compositional specification of ODP binding objects" – Proceeding of the INDC'96 conference, June 1996. Chapman & Hall.

FMOODS'96 First IFIP Workshop on Formal Methods for Open Object-based Distributed Systems, Paris, 4-6 March, 1996. Najm E. and Stefani J-B. Eds. Chapman & Hall.

Järvinen H-M., Kurki-Sunio R., Sakkinen M. and Systä K. (1990) "Object Oriented Specification of Reactive Systems" Proceedings of the 12th International Conference on Programming Languages and Systems, Nice, France, March 28-30, 1990. IEEE Computer Society Press.

Johnson D. R. and Kilov H. (1996) "Can a flat notation be used to specify an OO system: using Z to describe RM-ODP constructs" in (FMOODS'96).

R. Milner, J. Parrow, D. Walker (1992) "A Calculus of Mobile Processes: Parts I & II" - Journal of Information and Computation 100, p 1-77, 1992.

Najm E. and Stefani J-B. (1991) "Dynamic Configurations in LOTOS" – Proceedings of the IFIP Forte'91 Conference, Sydney, 1991. North Holland.

Najm E. and Stefani J-B. (1995) "A Formal Semantics for the ODP Computational Model, with signals, explicit binding and reactive objects" – CNET technical report no NT/PAA/TSA/TLR/4187, January 1995.

Najm E. and Stefani J-B. (1995) "A Formal Semantics for the ODP Computational Model" – Computer Networks and ISDN Systems, Volume 27, 1995.

Nierstrasz O. (1993) "Composing active objects" in (Agha et al. 1993).

Pierce B. and Turner D. (1995) "Concurrent Objects in a Process Calculus" Proceedings Theory and Practice of Parallel Programming, Ito T. and Yonezawa A. (Ed.), Springer Verlag LNCS 907, Sendai, Japan, 1995.

Prinz A. (1996) "Including Behaviour into Interfaces" in (FMOODS'96).

Rudkin S. "Inheritance in LOTOS". In K.R. Parker and G.A. Rose, editors, Formal Description Techniques, IV, (North Holland), 1992. Elsevier Science Publishers B.V.

Rumbaugh J., Blaha M., Premerlani W., Eddy F. and Lorenson W. (1991) "Object-Oriented Modeling Technique" - Prentice Hall, 1991.

RM-ODP (1995) ITU-T Rec. X.901-904 / ISO/IEC 10746 1-4: "Reference Model of Open Distributed Processing" – Parts 1-4 (Part 1: Overview, Part 2: Descriptive Model, Part 3: Prescriptive Model, Part 4: Architectural Semantics). – July 1995.

Stefani J-B. (1992) "Open Distributed Processing: The Next Target for the Application of Formal Description Techniques" – Proceedings of the IFIP Forte'90 Conference, Madrid. North Holland.

Stefani J-B. (1995) "Open Distributed Processing: an architectural basis for information networks" – Computer Communications, Vol. 18 - Num. 11, November 1995. Elsevier Science B. V.

Vogel A. (1993) "An ODP architectural Semantics using LOTOS" - Proc. 1st International Conference on ODP, Berlin, September 1993.

Vogel A. (1994) "A Formal Approach to an Architecture for Open Distributed Processing" - Technical Report #902, Université de Montréal (1994).

Wegner P. (1990) "Concepts and Paradigms of Object-Oriented Programming" ACM OOPS Messenger, vol. 1, no. 1, pp. 7-87, August 1990.

Joubine Dustzadeh received his Master's degree in Theoretical Physics from Ecole Normale Supérieure Ulm, France (in conjunction with the University of Paris VII) in 1988 and then graduated from Ecole Nationale Supérieure des Télécommunications ENST in 1990. From 1990 to 1993, he worked as a Software Engineer in various areas of telecommunications at Unisys and Retix Corporations in the United States. Since 1993, he is a Research Engineer at the Networks Department of ENST in the group of Formal Methods for Distributed Systems and Applications where he is preparing a Ph.D. within a joint research program between SITA (Société Internationale des Télécommunications Aéronautiques) and ENST.

Elie Najm has the grade of "Habilitation of Director of Research" from the University of Paris VI (1993) and has graduated from Ecole Nationale Supérieure des Télécommunications ENST in 1977 and Ecole Polytechnique in 1975 . He is currently Senior Lecturer at ENST and head of the DAF research group on Formal Methods for Distributed Systems and Applications.