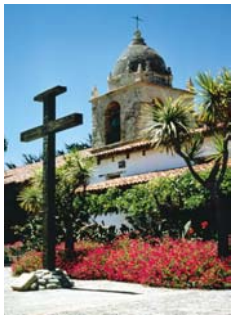
# Workshop on ODP for Enterprise Computing (WODPEC 2004)

A. Vallecillo, P. Linington and B. Wood (Eds.)

Monterey (CA), September 20, 2004
http://www.lcc.uma.es/~av/wodpec2004/

# Proceedings



In conjunction with:

**The 8th International IEEE Enterprise Distributed Object Computing Conference**
**20-24 September 2004, Monterey, California, USA**
**http://www.edocconference.org**

Sponsored by IEEE Computer Society,
IEEE Communications Society,
and the University of Alabama at Birmingham (UAB)
University of Málaga. ETSI Informática.
Dept. Lenguajes y Ciencias de la Computación.

**Technical Report No. ITI-04-07**

## Editors

**Antonio Vallecillo**
University of Málaga
ETSI Informática
Campus Teatinos
29071 Málaga (Spain)
av@lcc.uma.es
http://www.lcc.uma.es/~av

**Peter F. Linington**
Computing Laboratory
University of Kent
Canterbury
Kent, CT2 7NF (UK)
P.F.Linington@kent.ac.uk
http://www.cs.kent.ac.uk/people/staff/pfl/

**Bryan W. Wood**
Open-IT Limited
11 Wilton Court
Sheen Road
Richmond
Surrey TW9 1AH (UK)
Bryan.Wood@Open-IT.co.uk
http://www.open-it.co.uk

ISBN 84-688-8046-9


Impreso en España – Printed in Spain
September 2004

# Table of Contents

# List of Authors



# Program Committee

| | |
|---|---|
| David Akehurst | University of Kent (UK) |
| Jean Bérubé | Idigenic (Canada) |
| Jonathan Billington | University of South Australia (Australia) |
| Celso González | IBM (Canada) |
| Haim Kilov | Stevens Institute of Technology (US) |
| Lea Kutvonen | University of Helsinki (Finland) |
| Juliette Le-Delliou | EDF (France) |
| Peter F. Linington | University of Kent (UK) |
| Arve Meisingset | Telenor (Norway) |
| Joaquin Miller | X-Change Technologies (US) |
| Tom Rutt | Coast Enterprises, INCITS T3 IR (US) |
| Akira Tanaka | Hitachi (Japan) |
| Sandy Tyndale-Biscoe | Open-IT (UK) |
| Antonio Vallecillo | University of Málaga (Spain) |
| Bryan Wood | Open-IT (UK) |

# Preface

The present volume contains the papers selected for presentation at the EDOC 2004 Workshop on ODP for Enterprise Computing (WODPEC 2004), that aims to provide a venue where researchers and practitioners on the use of the RM-ODP in the realm of Enterprise Distributed Computing can meet, disseminate and exchange ideas and problems, identify some of the key issues related to these topics, and explore together possible solutions and future work.

All papers submitted to WODPEC 2004 were formally peer reviewed by at least three referees, and 9 papers were finally accepted for presentation at the workshop. These contributions can be divided into two groups: "RM-ODP basics" and "Tools and Applications". The first group contains the papers that deal with general issues of the RM-ODP. The papers in the second group deal with applications of the RM-ODP, tools to support it, and its relationship with MDA.

We would like to thank the EDOC 2004 organization for giving us the opportunity to organize the workshop, especially to the General Chair, Barret Bryant, and the Workshop Chair, Jishnu Mukherji, for their assistance and support. Many thanks to all those that submitted papers, and particularly to the contributing authors. Our gratitude also goes to the paper reviewers and the members of the WODPEC 2004 Program Committee, who helped in choosing and improving the selected papers. Finally, we want to acknowledge the Department of Lenguajes y Ciencias de la Computación of the University of Málaga for supporting the production and distribution of this volume, and the Spanish CICYT research project TIC2002-04309-C02 for funding some of its costs.

Monterey, California, September 2004

Peter Linington, Bryan Wood and Antonio Vallecillo
WODPEC 2004 Organizers

# Workshop on ODP for Enterprise Computing (WODPEC 2004)

Antonio Vallecillo[1], Peter F. Linington[2] and Bryan Wood[3]

*1 Universidad de Malaga, Spain; 2 University of Kent, UK; 3 Open IT, UK.*
*1 av@lcc.uma.es; 2 pfl@kent.ac.uk; 3 bryan@open-It.co.uk.*

## Abstract

*This Workshop aims at providing a discussion forum where researchers, practitioners and representatives of standardization bodies on these topics can meet and exchange experiences, problems and ideas related to the use of RM-ODP in the realm of Enterprise Distributed Computing, and explore together possible solutions and future work.*

## 1. Introduction

As software technology becomes a core part of business enterprises in all market sectors, customers demand more flexible enterprise systems. This demand coincides with the increasing use of personal computers and digital assistants, the trend of IT organizations towards downsizing, and today's easy access to local and global communication networks, which together provide an excellent infrastructure for open distributed systems.

In response to these market needs, there has been a significant development in International Standards in software and system engineering in the last decade. In particular, the rapid growth of distributed processing has led to the adoption of the Reference Model of Open Distributed Processing (RM-ODP, [1] [2] [3] [4]). This Reference Model provides a co-ordinating framework for the standardisation of open distributed processing (ODP) and creates an architecture within which support of distribution, interworking, and portability can be integrated, together with a framework for the specification of ODP systems.

The RM-ODP is based on precise concepts derived from current distributed processing developments and, as far as possible, on the use of formal description techniques for specification of the architecture. RM-ODP has four fundamental elements: an object modelling approach to system specification; the specification of a system in terms of separate but interrelated viewpoint specifications; the definition of a system infrastructure providing distribution transparencies for system applications; and a framework for assessing system conformance.

Five years after its final adoption as an ITU-T Recommendation and ISO/IEC International Standard, the RM-ODP is increasingly relevant, because the size and complexity of current IT systems is challenging most of the current software engineering methods and tools. These tools were not conceived for use with large, open and distributed systems, which are precisely the systems that the RM-ODP addresses

The RM-ODP has already lead to some real experience and supporting systems, including both good and bad reports; a number of conformant products and implementations; some related developments through the Object Management Group and, of course, groups both of supporters and of detractors. The knowledge gained from these experiences, together with the fact that ISO has just launched a Study Group to consider the possible revision of the Reference Model (see [16]), provides an excellent opportunity for all parties involved in Enterprise Distributed Computing to meet and discuss the present state and future development of the RM-ODP and its related family of standards.

In this context, this Workshop aims at providing a discussion forum where researchers, practitioners and representatives of standardization bodies on these topics can meet and exchange experiences, problems and ideas related to the use of RM-ODP in the realm of Enterprise Distributed Computing, and explore together possible solutions and future work.

## 2. The Reference Model

In slightly more detail, the current components of the RM-ODP are as follows:

**ITU-T Rec. X.901 | ISO/IEC 10746-1: Overview** contains a motivational overview of ODP giving scoping, justification and explanation of key concepts, and an outline

of the ODP architecture. It contains explanatory material on how this Reference Model is to be interpreted and applied by its users, who may include standards writers and architects of ODP systems. It also contains a categorization of required areas of standardization expressed in terms of the reference points for conformance identified in ITU-T Recommendation X.903 | ISO/IEC 10746-3. This part is not normative.

**ITU-T Rec. X.902 | ISO/IEC 10746-2: Foundations** contains the definition of the concepts and analytical framework for normalized description of (arbitrary) distributed processing systems. This is only to a level of detail sufficient to support ITU-T Rec. X.903 | ISO/IEC 10746-3 and to establish requirements for new specification techniques. This part is normative.

**ITU-T Rec. X.903 | ISO/IEC 10746-3**: **Architecture** contains the specification of the required characteristics that qualify distributed processing as open. These are the constraints to which ODP standards must conform. It uses the descriptive techniques from ITU-T Rec. X.902 | ISO/IEC 10746-2. This part is normative.

**ITU-T Rec. X.904 | ISO 10746-4**: **Architectural semantics** contains a normalization of the ODP modeling concepts defined in ITU-T Rec. X.902 | ISO/IEC 10746-2 Clauses 8 and 9. The normalization is achieved by interpreting each concept in terms of the constructs of the different standardized formal descriptions.

## 3. The ISO review process

The ISO Study Group is concerned with requirements for revision of the Reference Model of Open Distributed Processing (RM-ODP). Its objective is the elicitation and analysis of requirements for the revision of the RM-ODP, and the preparation of a set of recommendations to ISO/IEC JTC1/SC7 about the actions, if any, that are required.

The RM-ODP is the mature result of a large amount of technical effort, consequently the Study Group is not expected to propose major restructuring or change, but to correct errors and take account of recent developments in the industry, so as to maintain the broad scope of the standard. However, if it becomes apparent during the study that there is merit in considering the standardization of alternative frameworks based on different structuring principles, the Study Group may recommend the establishment of new projects to that effect. In proposing any update to the RM-ODP, the group will take into account the need to remain consistent with existing standards based on it, such as the Enterprise Language [6], the ODP Trader [10] [11], the Naming Framework [5] and the standard for Interface References and Binding [8].

The Study Group will prepare a report to present to the SC7 Plenary in May 2005. The report will document requirements submitted and the actions proposed on them. Recommendations on actions may include:

a) Proposing technical corrigenda to the existing standard;

b) Proposing one (or more) new projects for the revision of the current standard;

c) Proposing one (or more) projects for the standardization, or mapping, of alternative frameworks.

The approach of the Study Group can be summarized as follows:

a) A dissemination activity, where the existence and mandate of this Study Group will be publicized to as wide an audience as possible;

b) A contribution activity, where submissions clearly identifying the requirements, the rationale, and the proposed revisions are made;

c) A collection activity, where requirements will be elicited, generated and collected.

d) An analysis activity, where requirements will be analyzed, classified and prioritized

e) A planning activity, where the rationale for action is established and concrete steps proposed.

## 4. Workshop organisation

The Workshop is expected to have four main sessions:

- Session 1: Introduction.
- Session 2: RM-ODP basics
- Session 3: Tools and Applications
- Session 4: Workshop conclusions

The Introductory session will present the schedule, contents and objective of the Workshop, and will also serve to introduce the participants of the Workshop.

"RM-ODP basics" will cover the (4) papers that address general issues of the RM-ODP:

1. “Semantic interoperability: using RM-ODP to bridge communication gaps between stakeholders.” H.Kilov.
2. “What Foundations does the RM-ODP Need?” P. Linington.
3. “Action Templates and Causalities in the ODP Computational Viewpoint.” R. Romero and A. Vallecillo.

4. “The role of the RM-ODP Computational Viewpoint Concepts in the MDA approach.” J.P. Almeida, M. van Sinderen and L. Ferreira Pires.

"Tools and Applications" will cover the (5) papers that deal with applications of the RM-ODP, tool support for it, and its relationship with MDA:

1. “Applying Model-Driven Development to Business Systems using RM-ODP and EDOC” Y. Nagase, D. Hashimoto and M. Sato
2. “Architecting frameworks for specific applications with RM-ODP.” A.P. Gonçalves Serra, S.A. Vicente, D. Karam Jr and M. Martucci Jr.
3. “A Model-Driven Approach for Information System Migration.” R. Le Delliou, N. Ploquin, M. Belaunde, R. Bendraou and L. Féraud.
4. “Challenges for ODP-based infrastructure for managing dynamic B2B networks.” Lea Kutvonen.
5. “Proposal for a Model Driven Approach to Creating a Tool to Support the RM-ODP.” D. Akehurst.

These two main sessions will share a common structure, with each of the presentations followed directly by questions for clarification, and the session concluded by a general wrap-up discussion. The items to be identified during the presentations of the papers and the wrap-up discussion at the end of each session could include the following:

- Issues, limitations, and current problems of RM-ODP
- Requirements for modifications, enhancements, and developments of RM-ODP
- Issues for the Study Group
- Identification of future work e.g., tools, etc.

The last part ("Conclusions") will try to identify the results from the Workshop. The idea is that this part not only concludes the Workshop, but also is used to write down the final issues that the Study Group may undertake, and to outline future work, useful tools that may be developed, etc. The intention is that WODPEC participants should take something away with them after the Workshop, namely some agreed ideas, work to do, etc.

## References


[1] ITU-T Recommendation X.901 | ISO/IEC IS 10746-1, *Information technology - Open Distributed Processing – Reference Model: Overview*, 1996.

[2] ITU-T Recommendation X.902 | ISO/IEC IS 10746-2, *Information technology - Open Distributed Processing – Reference Model: Foundations*, 1996.

[3] ITU-T Recommendation X.903 | ISO/IEC IS 10746-3, *Information technology - Open Distributed Processing – Reference Model: Architecture*, 1996.

[4] ITU-T Recommendation X.904 | ISO/IEC IS 10746-4, *Information technology - Open Distributed Processing – Reference Model: Architectural Semantics*, 1996.

[5] ITU-T Recommendation X.910 | ISO/IEC 14771, *Information technology - Open Distributed Processing - Naming framework*, 1998.

[6] ITU-T Recommendation X.911 | ISO/IEC 15414, *Information technology - Open Distributed Processing-Enterprise Language*, 2001.

[7] ITU-T Recommendation X.920 | ISO/IEC 14750, *Information technology - Open Distributed Processing - Interface Definition Language*, 1997.

[8] ITU-T Recommendation X.930 | ISO/IEC 14753, *Information technology - Open Distributed Processing - Interface references and binding*, 1998.

[9] ITU-T Recommendation X.931 | ISO/IEC 14752, *Information technology - Open Distributed Processing - Protocol support for computational interactions*, 1999.

[10] ITU-T Recommendation X.950 | ISO/IEC 13235-1, *Information technology - Open Distributed Processing - Trading Function: Specification*, 1997.

[11] ITU-T Recommendation X.952 | ISO/IEC 13235-3, *Information technology - Open Distributed Processing - Trading Function: Provision of trading function using OSI Directory service*, 1997.

[12] ITU-T Recommendation X.960 | ISO/IEC 14769. *Information technology - Open Distributed Processing - Type repository function*, 1999.

[13] http://www.lcc.uma.es/~av/ODPStudyGroup/SGannouncement.html

# Using RM-ODP to bridge communication gaps between stakeholders

**Haim Kilov**
*Independent Consultant*, and *Stevens Institute of Technology*
*haimk@acm.org*

## Abstract

*The proverbial communication gap between business and IT experts has led to substantial problems in interoperability between different stakeholders of different (business and IT) systems, leading, in turn, to significant monetary losses together with loss of customers' trust and patience. The paper demonstrates not only the problems but also the solution — a clear separation between the business and IT domains based on an explicit usage of a system of concepts common to all domains and understood by all stakeholders. These elegant concepts come from exact philosophy, mathematics, programming and systems thinking and have been described in an international standard, the Reference Model of Open Distributed Processing The paper shows how a system of exactified concepts and approaches has been used to understand and specify the semantics of non-trivial industrial business and IT systems, thus establishing a basis for successful communication between business and IT experts, that is, for semantic (and sometimes syntactic) interoperability.*

## 1. Introduction

In thinking about and discussing interoperability, we observe that the systems that have to interoperate need not be computer-based. Specifically, we may — and probably ought to — look at business-based and IT-based stakeholders as components of several systems that often have serious difficulties in communicating.

The proverbial communication gap between business and IT experts has led to substantial problems in interoperability (both syntactic and semantic) between different stakeholders of different (business and IT) systems, leading, in turn, to significant monetary losses together with loss of customers' trust and patience. The paper demonstrates not only the problems but also the solution — a clear separation between the business and IT domains based on an explicit usage of a system of concepts common to all domains and understood by all stakeholders. These elegant concepts come from exact philosophy[1], mathematics, programming and systems thinking. They have been successfully used not only in theory but also in industrial practice, in international standards such as the Reference Model of Open Distributed Processing — RM-ODP, and in teaching of business and IT modeling. The paper shows how a system of exactified concepts and approaches, especially such concepts as system, type, relationship, composition, pattern, name in context, etc., has been used to understand and specify the semantics of non-trivial industrial business and IT systems, thus establishing a basis for successful communication between business and IT experts, that is, for semantic (and sometimes syntactic) interoperability.

## 2. Conceptual requirements for interoperability

Let us start with a description of two familiar kinds of interoperability. "Syntactic interoperability is all about parsing data correctly. Semantic interoperability requires mapping between terms, which in turn requires content analysis. This requires formal and explicit specifications of domain models, which define the terms used and their relationships. Such formal domain models are sometimes called ontologies." [32] This description may be used not only for dealing with computer-based information systems but also — and perhaps more importantly — for dealing with *human stakeholders communicating with other humans or with computer-based systems*.

This reference to domain models and relationships is not new. Walter Bagehot, one of the founders of modern money markets, suggested in 1873 the same approach in order to understand and communicate about the objects in the money world: "[t]he objects which you see in Lombard Street, and in that money world which is grouped about it, are the Bank of England, the Private Banks, the Joint Stock Banks, and the bill brokers. But before describing each of these separately we must look at what all have in common, and at the relation of each to the others." [2].

Recognition of *importance of domain models* has not been always forthcoming in the context of computer-based

[1]Our interest in and usage of exact philosophy may be explained, for example, by Bunge's observation that concepts and hypotheses are philosophical because they occur in a large number of fields of inquiry [4]. This was recognized in the curricula of (at least) the early universities. Business analysts as generalists who work on analyzing any systems, and (information) system designers belong to the esteemed class of people who use and sometimes develop such concepts and hypotheses.

information systems. However, in any engineering discipline, domain understanding and specification is essential before system requirements can be understood and formulated, and of course, a system can be developed only on the basis of well-understood requirements. This was emphasized, for example, by Dines Bjørner: "domain, requirements and software design are three main phases of software development". As we all know, sometimes IT system requirements exist only "in the collective heads" of the developers, leading to more or less serious failures described, for example, in Peter G. Neumann's "Risks to the Public" column in *Software Engineering Notes*. Therefore interoperability between stakeholders during domain modeling, as well as during requirements understanding and specification, is essential for success in development of any systems, including software systems.

The communication gap between business and IT experts can be exactified as the absence of interoperability. Firstly, often there is no *syntactic* interoperability because business stakeholders are often unable to parse data used by IT stakeholders. Of course, business experts cannot (and should not!) read code, but they also often cannot read specifications written by IT experts using notations[2] (or terminology) which are overly complicated or alien to business. Here is an example from the OOPSLA'99 keynote (on e-business) by Stu Feldman: "We need to understand the domain before addressing software. … Business models are the basis of an organization's entire activity. They are to be understood by CEO and CFO, not just by CIO; and therefore explained without 'method calls will have an XML representation'."

Secondly, often there is no *semantic* interoperability between business and IT experts and also between different business experts, as well as between different IT experts. This happens because the same data (such as "meaningful names") may be and often are interpreted differently by different stakeholders, and in the absence of an explicit domain model these differences may not be discovered until it is too late. Grace Hopper observed in 1957 that "[w]hile the computation of the square root of a floating decimal number remained the same in Pittsburgh, Los Angeles, and New York, the computation of gross-to-net pay obviously did not remain the same even in two installations in the same city" [9]. The same kind of problems still exists today, and we still often encounter statements like "everyone knows what a 'patient' is", or "everyone knows what a 'trade confirmation' is". Many tool vendors avoid these issues, and for a good reason: understanding and solving semantic interoperability problems requires human intervention that cannot be replaced with any tools.

## 3. Obstacles for interoperability

Interoperability problems are not specific to software systems and have been acknowledged by many business and IT experts. Solution attempts have existed for quite a while, but often they resulted in not much more than warm and fuzzy feelings during meetings. Many if not most failed attempts were based on various more or less fashionable information technology tools and methodologies instead of clear and explicit domain ontologies. On the one hand, most specifications have relied on (a lot of) tacit assumptions which are clearly different for different specification readers. On the other hand, even explicit fragments of specifications presented using box-and-line diagrams (as all too often various architectures and even business plans have been shown), or in natural language, lead to serious problems because different people will interpret the specification differently. Indeed, in order to transmit a message from one person to another without loss of meaning, the author and recipient(s) of the message have to use the same ontology and the same notation. Using the same natural (colloquial) language as a notation is not sufficient since in order to preserve meaning we need also the same context, the same language experience, language norms, cultural tradition, and so on [21], and these properties of different people are often implicit and (very) different. At the same time, a restrictive artificial language *with precisely defined semantics* that does not have contexts, cultural traditions, and so on, can guarantee an adequate transmission of a message's *semantics*, provided, of course, that it can be adequately represented in that language. As a somewhat crude approximation of such a restrictive language, we may consider "legalese" in which, for example, laws (providing "the same context") and contracts are written.

More recently, the recognition of importance of ontologies (see, for example, an overview in [37]) could have become a much-needed (social) innovation — not even a radical one — used to solve the interoperability problems.

However, ontology development today is in a poor state. Often, it has been replaced with an emphasis on (a large number of) logic and ontology languages, in the same manner as programming has been replaced with an emphasis on various "baroque programming languages" blurring our vision "by the wealth of their mutually conflicting 'powerful features' " (Dijkstra), or in the same manner as analysis has been replaced with an emphasis on "Undefined modeling languages" (Parnas). As a (or the) result, the complexity of the domain and problems has been replaced with the complexity of the language (as

[2]Business experts have no time or desire to study — for 5 days, 8 hours a day — the many facilities of a notation using toy examples, as often happens when popular "powerful" notations are taught.

Dijkstra observed, such languages were used to express in a funny way the usually given algorithms). Furthermore, in the context of modeling and ontology languages, methodologies are being created for the sake of understanding how to use the usually complex tools. Because of the lack of any generally accepted processes and methodologies, the tools exist independently and have little support for or concern with interoperability (Ken Baclawski). Also, most conceptual modeling activities have proceeded without the benefit of theory [37]. These often buzzword-compliant approaches are tinkering (Bunge) rather than engineering ones.

The primary goal of a programming language is accurate communication among humans. Clearly, the same is true about a modeling (or ontology) language. With many currently popular languages, this goal has not been reached.

As a result, even in cases when a specification exists and has been agreed upon by the relevant stakeholders, it may not guarantee interoperability of compliant systems, computer-based or otherwise. For example, if one vendor of a supposedly compliant product interprets the specification one way and another vendor interprets it another way, then both will claim compliance yet the products won't interoperate, and nobody can say for certain which interpretation is "the right one". For another example, different business experts may agree on the apparent validity of the business specification, but may interpret tacit underlying assumptions in different ways, and therefore their understandings of the specification — composed of the explicit parts and the tacit assumptions — will differ leading to serious (IT and) business problems. As an illustration, we may recall the "dot-com" epoch when people "suddenly realized that they had invested a fortune based on a few beautiful graphics that were laughably called a business plan" [22].

## 4. A system of common concepts: The way to address obstacles to interoperability

### 4.1. The need for effective patterns of reasoning

Information management systems are becoming more complex and non-trivial since they have to serve the needs of complex, non-trivial and rapidly changing businesses. In order to succeed in understanding, specifying, designing and developing information systems we should do better than use rigid methodologies combined with step-by-step approaches, or, alternatively, specification-free approaches[3].

This appears to be difficult (but is not) and perhaps unusual. As E.W.Dijkstra observed a while ago, "many students don't want to be shown effective patterns of reasoning, they want to be told what to do. ... They expect a so-called 'complete methodology'... and complain when they don't get what only the quack can provide. (We just addressed a bunch of industrial computing scientists, and the above phenomenon was alarmingly pronounced.)" [5]. Instead of using "junk food" — the metaphor for simplistic methodologies we owe to Paul Clermont — we should better use effective patterns of reasoning that help at all stages of information management, as well as in business management, independently of any possible computer-based realization of its fragments.

### 4.2. Where do we get these patterns of reasoning?

We certainly do not *want* to invent them for each and every project or stage of a project. Fortunately, we do not *need* to do that either: an excellent and well-structured system of common concepts on which patterns of reasoning may be based, already exists. It was defined in an abstract, precise and concise — elegant! — manner as an international standard: the *Reference Model of Open Distributed Processing (RM-ODP)* standardized by the International Standards Organization (ISO) in 1995 [25, 26].

### 4.3. Semantics

RM-ODP specifies semantics in a manner that is syntax-, methodology-, and tool-neutral. It provides *precision without programming*[4]. The Foundations of RM-ODP are very short — only 18 pages. Every concept there is precisely defined in clearly structured English within the context of other precisely defined concepts. In other words, the reader

[3]Here is a fragment of the description of the event "Using formal methods to understand requirements better" at the Imperial College London (November 6, 2002) [27]: "**Anthony Hall** : [...] by being precise, early, you could discover problems and reduce trouble during development.[...] Precise requirements could be defined in Z, a logic notation. [...] Z found as many errors as unit testing, but was five times cheaper. It found errors early too: many of the errors introduced by specification were removed during architecture or detailed design, and in a recent project only one specification error survived through to operations. Modestly but with some pride he quoted an authentic customer statement: "The system behaves impeccably as expected." **Axel van Lamsweerde** said that he agreed 1000%, but what of people who advocated agile methods? Anthony Hall replied by quoting John Barnes: "Ada is not meant to make programming quick. It's meant to make programming slow. Slow is good". There was laughter."

[4]This expression was used by Anthony Hall. Some IT experts claim that "business people cannot understand precision" and that therefore business experts should be provided only with various narratives and pictures instead of precise specifications. These condescending claims are obviously wrong since business experts have understood precision and made precise decisions for millennia.

does not have to figure out what a particular term means, and neither does the reader have to rely on tacit assumptions left undefined since "everyone knows what this means" (but do they know and mean the same things?).

RM-ODP eliminates much of the work that would otherwise be required by each organization to develop its own similar but proprietary guidelines — patterns of reasoning used to understand and to specify businesses as well as to specify, design and develop information systems for these businesses. RM-ODP (hopefully) will also provide the incentive to many business and IT organizations to follow similar approaches in developing specifications, leading to industry standards. RM-ODP has already been substantially used in creating other important standards, such as ISO standards — General Relationship Model and Trader, and OMG standards — UML Profile for Enterprise Distributed Object Computing, Model-Driven Architecture, and others, as well as industry-specific (vertical) business-specification standards.

### 4.4. Is this a radical novelty?

There is nothing radically new here. The need to elucidate the definitions of things, actions, and especially the *structure (relationships)* of a system for understanding of that system has been noted by many authors, both in modern-day information management and systems thinking, and much earlier (recall the quote from Bagehot, for example). Moreover, the concepts essential to understand and specify the semantics of system components and structure have been formulated and discussed in IT, mathematics, philosophy, and system analysis for a while. RM-ODP and the international standard used to describe relationships in more detail — the General Relationship Model (GRM) [8] — define a *system* of these concepts and are based on these ideas. This system includes such concepts as system, abstraction, viewpoint, level, object, action, state, behavior, type, subtype, template, composition, refinement, contract, name, context, invariant, pre- and post-condition, failure, error, and conformance. These *semantics* concepts are not associated with a particular technology approach (such as object orientation) and are neutral with respect to representational or tool-related issues.

Many of these concepts are quite familiar to good programmers and analysts. In particular, most have been around in programming since the mid-1960s. Many have also been successfully used in various modeling approaches within the framework of the three-schema database architecture. Some of them have been used in engineering and other areas of human endeavor (such as business and law) for centuries. The RM-ODP definitions are theoretically sound (based on mathematics, as demonstrated, for example, in the Architectural Semantics part of the standard) and have been successfully used in practice. In the same manner as businesses in the US rely on the standard Uniform Commercial Code, system specifiers ought to rely on the standard RM-ODP.

### 4.5. Is RM-ODP really useful?

Here is an assessment of using RM-ODP to specify systems (from the European Air Traffic Management System Architecture Workshop, held at EUROCONTROL headquarters in Brussels on June 11-13, 1996): "The application of RM-ODP provided insight into a number of interoperability issues... The RM-ODP... quickly showed the importance of correct focusing, scope, interpretation and representation within the descriptions. The RM-ODP approach impels the analyst to state clearly the focus and scope at the beginning of the process... The model provides a common, consistent and incremental approach for describing the goals, objectives and behaviour of systems in detail. Hence, it offers support for life-cycle management and for strategy studies. In a similar way, it also offers support for COTS procurement – both for the procurement specification and for the suppliers' system specification and design specification."

Look at this evidence. Not only was RM-ODP useful at the specification stages; it was valuable at all stages of information management including procurement and evaluation of suppliers' systems. This is especially important in environments where "virtually all users are configuring systems, not developing them from scratch" [28].

There exists a lot more evidence of this kind, not only in air traffic control but also in less exotic areas of telecommunications, finance, insurance, document management, medical, pharmaceutical and other industries, as well as in management and strategic consulting. For example, RM-ODP (together with GRM) was used:

- to elucidate and describe various architectures in a large international financial institution,
- to provide a successful communication mechanism for stakeholders in business process reengineering projects (and to describe these projects from specification to realization),
- to specify COTS software components in a simple and understandable manner so that the semantics of these components became clear to their users,
- to create and use simple and elegant (and complete) business specifications of various financial domains (such as accounting, trading, exotic options, etc.) used by large financial firms in their work, both for information system creation and for business process change,
- to specify products by healthcare software vendors

transforming creation of these specifications "from craft to a formal science",
- to develop a complete human resources model for DoD,
- to formulate a clear model of document management separating the concerns of content, logical design, and physical presentation,
- to provide a foundation for business decisions related to mergers and acquisitions,
- to elucidate and formulate fragments of the UML metamodel,
- to describe business strategies,
- to describe various business patterns,
- to develop ontologies of various business domains,
- to build the industry library in the pharmaceutical industry,

and so on. Some of these applications were described by satisfied clients in literature [16 (several papers), 6, 18, 13, 10, 15, 7, 31, 23, 34, 24].

### 4.6. Separation of concerns based on a common foundation

RM-ODP makes it possible (although not trivial!) to formulate understandable specifications in a disciplined manner. Specifications will be read by people who are non-experts in specifications. This especially applies to *business specifications* [11, 12, 14] all too often reduced to the elusive "business rules" (which are "in the code").

Discipline means precision and abstraction. *Precision* means (among other things) that a developer will not have to invent business rules that have not been described at all or have been described in an ambiguous or incomplete manner. *Abstraction* means (among other things) that a subject matter expert will not waste time and effort trying to understand business rules in terms of a particular computer-based implementation. Business rules (and a business enterprise in general) should be specified using abstract and precise concepts understandable to any good subject matter expert, analyst, developer, or (even) a non-IT manager. (Recall the quote from Stu Feldman, above.) A system of these concepts is the same for all kinds of specifications — thus providing *an excellent foundation for interoperability* — and has been formulated and described in RM-ODP and standards based on it, such as GRM.

The basic concepts defined in RM-ODP and GRM can be — and have been — used not only to describe any kinds of traditional businesses, but also to describe the essentials of any existing or to-be-created information systems (computer-based or not). In this manner, the business and IT stakeholders are able to use a common system of concepts and therefore to communicate in a meaningful manner. Of course, the syntactic representations used by different stakeholders to represent the same semantics may and often does differ, and also of course, different stakeholders may be interested in different levels and viewpoints when describing the same system, but the underlying semantic framework still remains the same.

While using the same system of concepts for all kinds of specifications, we should explicitly separate business from IT system specifications because traditional business and IT ontologies are different. (As a well-known example, a patient is not the same as one of patient's records.) Within each specification we should separate concerns as soon as we see that the specification (including a program — a specification for a computer system) becomes too complex for human understanding and is in danger of having "too much stuff". ("Precise" is not the same as "detailed", and therefore being abstract does not mean being imprecise. Good specifiers, in the same manner as good engineers, postpone decisions and do not get drowned in details. The higher the level of abstraction the more important it is to be precise!)

## 5. Well-structured specifications

### 5.1. Relationship semantics

A specification should be well-structured since only in this manner it can be carefully considered, read and understood. The structure of a system is "the collection of relations among its components or among these and items in its environments" [4]. Therefore precisely defined *relationship semantics* is essential for reading, understanding, and creating any specification. The semantics of a relationship is defined by means of its *invariant* referring to (collective) properties of relationship participants. Fortunately, it has been possible to specify the structure of a large number of diverse business and IT systems — such as financial derivatives, insurance underwriting, telecommunications systems, document management, UML metamodels, messaging, and IT system architectures — using only three kinds of generic relationships: *composition*, *subtyping* and *reference*. Semantic definitions of these relationships have been around for some time (see, for example, the exactifications in RM-ODP and GRM, as well as the modeling texts [19, 11, 14]), and it is important and instructive to observe that these definitions are based on *property determination*. In particular, the existence of emergent properties of a composite is the defining characteristic of the composition relationship[5] (see [11, 14, 4]

[5]A composition is a relationship between a "whole" (composite) and its "parts" (components). The type of the whole corresponds to the

for many examples). Thus, it also becomes blindingly obvious why an often encountered statement "this [named] line between these two boxes formally represents the relationship between these two things" does not convey anything at all about the relationship semantics, and therefore why box-and-line diagrams are inadequate for understanding and decision making.

### 5.2. Abstraction levels and viewpoints

Precision (exactification) is not sufficient for human understanding. Indeed, hundreds or thousands of pages of precise material are (almost) useless if the material is not well-structured. In other words, understanding requires abstraction — "suppression of irrelevant detail" [25], so that essential (for a specific level or for a specific viewpoint) aspects of a specification are clearly separated from accidental ones. Clearly, this is not a new approach, but it has not been stressed in most syntax- , methodology- or tool-oriented texts. And it is instructive that the concepts of abstraction, levels and viewpoints are among the first ones to be described in RM-ODP.

RM-ODP uses abstraction within the context of *levels*. In particular, it notes that "fixing a given level of abstraction may involve identifying which elements are atomic". For a more specific example, the concept of composition is defined using abstraction levels: "[a] combination of two or more [items] yielding a new [item], at a different level of abstraction. The characteristics of the new [item] are determined by the [items] being combined and by the way they are combined."

RM-ODP also uses abstraction within the context of *viewpoints* — "form[s] of abstraction achieved using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within a system". All viewpoints are based on the same system of basic concepts. RM-ODP specifies five basic viewpoints — enterprise, information, computational, engineering, and technology. It is possible to define correspondences between viewpoints (RM-ODP shows how to do that and provides some examples), but often, one viewpoint cannot be defined in terms of another. Of course, the five basic viewpoints are not the only ones that may be used to describe a system. In accordance with the definition of a viewpoint, any reasonable set of architectural concepts and structuring rules may be chosen to focus our attention on particular concerns within a system; and therefore we often use a *business viewpoint* and an *information system viewpoint*[6]. In this manner, for example, we can exactify the slogan "no requirements in terms of solutions" since requirements and solutions belong to different viewpoints.

types of the parts, and an instance of the whole corresponds to zero or more instances of each type of the part. There are two kinds of the properties of the whole those that are determined by the properties of the parts and the way these parts are combined; and those that are independent of the properties of any parts. A composition also satisfies the general relationship invariant that implies, in particular, that an instance of the whole cannot have itself as a part. This definition is based on the definition of composition in RM-ODP, GRM, on the definition of composition used in systems thinking (e.g., by F.A.Hayek) and in exact philosophy e.g., by Mario Bunge), and on usage of composition by such classics as Adam Smith in his *Wealth of Nations* (see [14]).

## 6. Business patterns and modeling

### 6.1. Only primitives?

RM-ODP provides a small but powerful system of interrelated definitional primitives that you can use to build your own specification. These primitives drastically reduce the number of base things and relationships and, hence, the complexity and size of a specification. More importantly, they *reduce the number of concepts* that the readers of a specification have to master in order to understand it.

The primitives need to be expanded in actual specifications. RM-ODP helps here in the form of "structuring rules" specifically designed to allow RM-ODP primitives to be used to develop more complex and/or specific definitions of various *business patterns*. These specialized definitions can be successfully mixed with the original primitives to create increasingly rich systems of definitions. This is similar to the way in which mathematicians create arbitrarily rich theorems from other theorems and well-understood basic axioms, or similar to how engineers create arbitrarily large and complex structures from common subassemblies.

The business patterns, of course, have to be discovered and explicitly formulated — and this is what business domain modeling is for. It discovers and specifies *deep analogies between seemingly different things, relationships, and processes*. In this manner, organizations can understand and deal with "always-changing" requirements as variations of a small number of conceptually simple patterns, leading to substantial savings in intellectual effort, time, and money. Among other things, it becomes possible to be demonstrably *proactive* rather than reactive in solving business problems because a clear and crisp business model may by itself provide a substantial competitive advantage for the modeled enterprise. To quote a satisfied client from a large financial firm, "[i]t has changed the way the client views software development and this single effort will serve as the foundation for other planned software development initiatives.

[6]This distinction between business and system viewpoints was made explicit, among others, in the distinction between computation-independent and the other two (platform-independent and platform-specific) viewpoints of the OMG's Model-Driven Architecture.

This business specification, written for software development, has potential application in other areas. Portions of the specification can be incorporated in corporate policy manuals; regulatory compliance documents, and serves as a basis for business process review." [7] In other words, various business and IT decisions could be based on a solid and explicit foundation rather than on handwaving, eloquence of gurus, or lemming-like considerations.

### 6.2. Patterns published

The reference list includes books and papers with fragments of business and IT system specifications based on RM-ODP and GRM. None of these are toy examples. Some of them are illustrative and fun but not trivial — like those modeling fragments of domains described by Lewis Carroll — while others are fragments from the generic parts of industrial specifications created for (and with active participation of) business and IT customers. These generic business and IT specifications may be, and have been, reused as business patterns in various customer engagements: after all, *pattern matching in context* is an essential part of successful analysis (and design). Since generic business patterns — such as, at different levels of genericity, invariant, composition, or contract— can be used in any application area, a good analyst can become a contributor in an entirely new area (and successfully interoperate with its stakeholders) within a very short timeframe. The conceptual foundation together with the generic business patterns let the good analyst to ask proper questions and "begin speaking the language [of the entirely new area] competently within a week or so" [35]. Thus, we may contrast Bunge's use of Claparede's criterion of intelligence as the ability to solve new problems [4] with an unfortunately typical help wanted advertisement for a business analyst stating that "knowledge of XXX is a must"

Business-specific business patterns can be discovered and formulated by reusing and exactifying their existing definitions, especially since some of them have been around for centuries, see, for example, Adam Smith's *Wealth of Nations*, rather than by rewriting and redefining them as "requirements" for each project. In other words, ontology reuse — and concept reuse in general — is much more valuable than code reuse.

Let us recall that Peter Naur proposed in 1968 [29] to use the work of Christopher Alexander long before it became fashionable to refer to it as a source of ideas about attacking the software design problem. Naur justified his choice by the fact that Alexander was concerned with the design of large heterogeneous constructions. Indeed, Alexander emphasized in *The Timeless Way of Building* that "...a pattern defines an invariant field which captures all the possible solutions to the problem given, in the stated range of contexts... the task of finding, or discovering, such an invariant field is immensely hard... anyone who takes the trouble to consider it carefully can understand it... these statements can be challenged because they are precise" [1].

Creating and elucidating a domain model cannot be automated. There is no algorithm (or tool) to do that. Such models are created and elucidated by teams consisting of domain SMEs (subject matter experts) and analysts. Even when the SMEs are experienced in formulating ontologies of their domain in an abstract (that is, understandable) and precise manner, analysts are still essential to discover, elucidate and exactify tacit assumptions common to all, or some, SMEs. *Ignorance* — real or perceived — of the subject matter is needed to make the tacit assumptions explicit. As early as 1969, P. Burkinshaw urged: "Get some intelligent ignoramus to read through your documentation; [...] he will find many 'holes' where essential information has been omitted. Unfortunately intelligent people don't stay ignorant too long, so ignorance becomes a rather precious resource." [30].

It is not sufficient to discover, formulate and use (fundamental, basic, and more specific) concepts and structures essential for a good model. We ought to communicate these discoveries, both for understanding of that model and for their usage in other, often apparently very different, models. In the modeling context, a concise and elegant system of basic concepts described in RM-ODP provides a foundation for such a language. Sometimes the *specifics* of this language or, more often, its fragments ought to be created (collectively, by the modelers together with the subject matter experts) for successful communication of a model's semantics. Models formulated in such a manner (of course, based on concepts and structures common to most systems) establish a common background used by all stakeholders of an organization and its relevant environments (e.g., clients and subcontractors) in understanding and business decision making.

## 7. Ontologies and invariants

### 7.1. Domain semantics

When we want to use an existing system (component) or plan to use a new one, we need to specify (for existing IT systems it often means "reverse engineer") the semantics of its interfaces, since only precise and explicit semantics make interoperability possible. This applies to any kind of system, independently of whether it is (or will be) computer-based. Clearly, in order to understand and define (the requirements for) interface semantics, we need to be *explicit*

and precise about the business domain in which the system works or will work. This is true for all kinds of businesses — be they traditional ones such as trading, or the business of creating an information management system, or of creating and using a relational database, or of asynchronous messaging, or of a particular general ledger legacy program. In other words, understanding and documenting of "what is there in the business domain" (also known as ontology) comes first. Within this explicitly specified framework, we can discuss the systems (and their interfaces) that work or are planned to work in this domain (after all, the descriptions of these systems refer to the things, relationships and actions of the domain!). And a data dictionary is *not* such a framework: as noted by many and as we know from practice, the same name may mean substantially or somewhat different things in different contexts, and the *structure* of these contexts — expressed in the relationships — is essential for understanding of the things we deal with.

Although ontologies have become fashionable rather recently, the underlying concepts for ontology understanding, development and use in business and IT system analysis are not new. They have been known from exact philosophy (notably, Bunge's work [3, 4]), database, object and information modeling work (for example, [37, 19]), systems thinking (F.A.Hayek[7], von Mises and others), and, of course, such international standards as RM-ODP and GRM. In addition to various kinds of subtyping, the concept of composition is among the most important in domain modeling, and its definition — based on levels of abstraction and on emergent property determination — is essentially the same in RM-ODP, GRM, and in Bunge's work.

### 7.2. The importance of elegance

With the introduction of standards like RM-ODP and GRM, today the analysis situation resembles the one that existed in programming in the second half of the 1970s. At that time, in E.W. Dijkstra's words, programming was in the process of moving from a craft to a scientific discipline. Most observations made by Dijkstra then for programming are reinvented now for analysis. We also see that in analysis, as in programming (an observation made by Dijkstra as early as 1962), elegance is of utmost importance; elegant specifications are liked by all and, thus, successfully used and reused. As Dijkstra said, "in the design of sophisticated digital systems, elegance is not a dispensable luxury but a matter of life and death, being a major factor that decides between success and failure", and a good specification ought to be convincing in the same manner as a good program is. For a specification to be of use (and to be produced in a reasonable manner, as any artefact produced by professional engineers), it needs to emphasize semantics rather than syntax, and to do that in an abstract and precise manner. This is precisely what RM-ODP and GRM have been designed to do.

### 7.3. "Learn to abstract: try not to think like a programmer" (J.Wing)

Both in programming and in analysis, the fashionable approaches have often encouraged practitioners to start their work in the middle using an operational approach. Jeannette Wing provided an excellent example of an approach to be avoided: "'If you do this and then that and then this and then that, you end up in a good state...' This [...] process quickly gets out of control. The problem is related to understanding invariants." [36]. *Invariants* are essential for defining the ontology of the domain of interest, be it a business or an information system one. In particular, they define the types of things, actions and relationships. They also determine what actions (and under what circumstances) can and cannot be executed in the context of the domain ontology. To quote a recent eloquent paper by Turski, one of the founders of computing science, a fundamental breakthrough in programming happened when "[i]nstead of preoccupation with a dynamic process ('what happens next'), we concentrated on a piece of text ('what does it say')" [33]. Indeed, the same kind of difference exists between various buzzword-compliant operational approaches to analysis that quickly get out of control, and elegant approaches that lead to understanding of businesses and information systems.

These elegant approaches start with and are based on ontologies. We do not want to start in the middle (that is, with a possible solution, as it too often happens) or even with a specific problem (that is, with requirements). Rather, *we start with the stable (that is, invariant) basics* and proceed from there. (The Information viewpoint of RM-ODP explicitly states that static and dynamic schemata are "subject to the constraints of any invariant schemata" [26].) Clearly, a problem and its solution cannot be understood and specified without the basics because the interrelated concepts used in describing the problem and the solution are defined by (and in) the basics.

### 7.4. Discover different ontologies

Substantial and especially somewhat different ontologies of different stakeholders (especially tacit ontologies) may make semantic interoperability difficult to achieve. In order to solve this problem, the existence of different ontologies

[7]For example, in accordance with Hayek's observations, prices constitute an essential emergent property that makes possible the functioning of a market economy and that embodies more information than each participant of a market economy directly has.

has to be made explicit. Ontologies — business domain models — are the framework for interrelating the existing vocabularies of different stakeholder communities instead of throwing these vocabularies away, thus providing interoperability solutions that really work, because they involve business *meanings*, while purely technical (or syntactical) solutions fail. Explicit ontologies make it possible to discover and specify mappings between concepts and relationships used in different ontologies (and often to enrich some of them). As a result, different stakeholders can communicate and interoperate: it will become clear which different context-specific terms have the same meaning and which identical context-specific terms have different meanings.

### 7.5. "Requirements always change"?

In this context, it is instructive to elucidate the (all too familiar) statement "requirements always change, and therefore it is useless to formulate them". Indeed, business *processes* often change. Such changes may lead to a competitive advantage for a business or they may even be perceived as necessary for the business to survive. Similarly, decisions about using IT systems to automate certain business processes may also change (for example, due to perceived opportunities). At the same time, the basics of a business — its ontology — have usually remained the same for centuries, if not for millennia: for example, banking and financial texts published in the early 20th century (or earlier, such as fragments from Adam Smith's *Wealth of Nations*) have been successfully used to understand and specify the corresponding business *domains*. The changes due to modernity are minimal and are mostly additions to or refinements of the existing classical models.

These considerations apply to any kind of business modeling as well as to requirements discovery and specification, independently of whether a computer-based IT system will be created or bought to automate some business processes or steps. As noted earlier, a crisp business model is used *to make demonstrably effective business decisions* only some of which are IT-related. And the concepts, constructs and standards used in creating such models are based on mathematics, "the art and science of effective reasoning" (Dijkstra).

## 8. Notations

Both in traditional programming and in modeling, we know from the works of Dijkstra and other founders — as well as from the experience of the best practitioners — that the inherent complexities of the problems and their solutions should not be exaggerated by imposing on the readers of programs or specifications a complex notation "with a plethora of *ad hoc* facilities of dubious value and unquestionable ugliness" [33]. In traditional programming, program readers and writers are usually humans with roughly the same background in the notation used. Contrariwise, in modeling (and specifications — the result of modeling), readers and writers often have drastically different backgrounds in the notations used. Therefore in order for business experts to use a precise notation to communicate about business models, it is absolutely essential to explain the basics of the notation on the 'back of an envelope'. This is important to get and retain the attention of the business people. Imposing a complex notation will move the reader's (and modeler's) attention from the complexity of the problems to the complexity of their representation. In other words, the attention will be moved from the essential semantics to the accidental syntax. As a result of such semiotic pollution, communication about problems suffers. This explains why many business system specifications are write-only, at least from the viewpoint of their main customers — the business decision makers.

Business (and any system) modeling notations should not be restricted by the artefacts existing or easily implementable in currently used IT systems. Thus, a modeling notation ought to be able to express multiple and dynamic subtyping, multiple decompositions of the same individual (thing or action), non-binary relationships with well-defined semantics, and other concepts defined in RM-ODP and GRM. For example, it should be possible to specify that a banking industry is a composite in a composition of banks, a federal regulator, customers, and a lot of something else (like clearing houses), and to say that in this composition both the composite and its components ought to exist together.

Turski stresses the need to limit ourselves in the process of understandable program construction to the systematic use — known as "structured programming" (invented by Dijkstra) — of a small collection of programming language instructions having "a clean and well-defined meaning". Further, he emphasizes that a high quality specification has to have a model not only in the programming language domain, but also in the language "used for the description of (a part) of the reality of interest", that is, understandable to the business subject matter experts. Following the ideas of structured programming, we may want to limit ourselves in the process of understandable specification construction to the systematic use of a *small collection of modeling constructs having a well-defined meaning*. And this collection, in fact, this system, already exists and was described in RM-ODP.

When we have to choose or use a specific notation or tool in our programming or modeling activities, we should, first and foremost, look at whether and how the semantics

of concepts we use in programming or modeling is supported by the tool. If the notation or tool is overwhelming then everything need not be lost: it may be possible to choose a (very) small subset of that notation in order to represent concept semantics. This approach is not new at all: it is well-known, for example, that various small subsets of PL/I have been created and used exactly for this purpose. Similarly, a very small subset of UML for business modeling [14] has been created and used in various engagements such as those described in [7, 12] and in several papers in [16], as well as for specifying relationships in [6]. This subset has been represented on one page.

At the same time, an important caveat is in order. When a subset of a notation is being chosen to represent concept semantics, it is essential for the semantics to have an exact representation in the notation. Since many important aspects of UML semantics have not been well-defined, it became necessary to provide for precise definitions of UML constructs used to represent the semantics of such concepts as composition, subtyping and reference relationships. This approach was accepted by OMG in the UML profile for EDOC [6].

## 9. Conclusions

It is now well understood that attempts to comply with a specification of any system having incomplete or unclear semantics will not guarantee interoperability because important information will be lost. The system of concepts defined in RM-ODP makes it possible to completely and precisely define the essential aspects of the universe of discourse, be it to describe a business or an information management system. These specifications are based on the semantics of the appropriate domain rather than on existing products or solutions. Since the system of concepts and constructs used for these specifications has been itself defined in a clear and crisp manner, the specifications can be *read*, understood, and thus agreed or disagreed upon by all stakeholders. Moreover, specifications of existing products or systems, including legacy systems, can and should be based on the same approach thus leading to demonstrably justified user decisions about acquiring such systems.

An RM-ODP-based specification provides a top-level precise (not semi-precise!) road-map of the appropriate fragment of a business or IT system, or of a product, and with its refinements down to the level(s) we are interested in. These specifications define what should always be true about the things and relationships of the business or IT domain as well as what should be true about each process (step, operation). All defaults are made explicit: in particular, the business expertise does not disappear when the business expert leaves the meeting room, and everything the developers need to know about the business domain and "were afraid to ask" is in the specification. Such specifications may be, and have been, used for making demonstrably justified strategic, tactical, and operational decisions in all kinds of business and IT system environments.

## 10. Acknowledgments

Many thanks go to the colleagues and customers the interactions with whom helped to distill the concepts and approaches described here. Some of their names are in the reference list below, and many names are in books and papers from this list. Also, specific thanks go to anonymous reviewers for very useful comments and to Rashid Bashshur for useful discussions about the structure of the paper. A draft of this paper was presented at the Telemedicine symposium in Ann Arbor, Michigan, in May 2004.

## 11. References

[1] Christopher Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

[2] W. Bagehot. *Lombard Street: A Description of the Money Market.* Scribner, Armstrong & Co., New York, 1873.

[3] Mario Bunge. *Philosophical dictionary.* Prometheus Books, 2003.

[4] Mario Bunge. *Emergence and convergence.* Toronto University Press, 2004.

[5] E.W. Dijkstra. *Management and Mathematics.* EWD966, The University of Texas at Austin, 14 June 1986.

[6] A UML Profile for Enterprise Distributed Object Computing Joint Final Submission Part I. 18 June 2001. OMG Document Number: ad/2001-06-09. 3.6. The Relationship Profile.

[7] James S. Garrison. Business specifications: Using UML to specify the trading of foreign exchange options. *Proceedings of the 10th OOPSLA workshop on behavioral semantics (Back to Basics)* (Eds. K. Baclawski and H. Kilov). Northeastern University, Boston, 2001, pp. 79-84.

[8] GRM. ISO/IEC JTC1/SC21, Information Technology. Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 7: General Relationship Model, 1995. ISO/IEC 10165- 7.2.

[9] Grace Hopper. *Automatic Programming for Business Applications.* In Proceedings of the 4th annual computer applications symposium, October 24-25, 1957, Armour Research Foundation, Chicago.

[10] J.Hassall, J.Eaton. Applying ISO RM-ODP in the specification of CORBA interfaces and semantics to general ledger systems. *Behavioral specifications of businesses and systems* (Ed. by H.Kilov, B.Rumpe, I.Simmonds), Kluwer Academic Publishers, 1999, pp. 91-104.

[11] H. Kilov. *Business Specifications*. Prentice-Hall, 1999.

[12] H.Kilov. *Anticipating the market: The value of business models.* Distributed Enterprise Architecture Advisory Service Executive Report. Cutter Consortium, 2001.

[13] Thomas Kudrass. Coping with semantics in XML document management. In *Proceedings of the 10th OOPSLA workshop on Behavioral Semantics — Back to basics.* (Tampa, Florida, 15 October 2001), pp. 150-161.

[14] H.Kilov. *Business models.* Prentice-Hall, 2002.

[15] H.Kilov, A.Ash. How to ask questions: Handling complexity in a business specification. *Proceedings of the OOPSLA'97 Workshop on object-oriented behavioral semantics* (Atlanta, October 6th, 1997), ed. by H.Kilov, B.Rumpe, I.Simmonds, Munich University of Technology, TUM-I9737, pp. 99-114.

[16] H.Kilov and K.Baclawski (Eds.) *Practical foundations of business system specifications.* Kluwer Academic Publishers, 2003.

[17] H.Kilov, M.Guttman. *ISO Reference Model for Open Distributed Processing: an informal introduction.* Cutter Consortium (Distributed Computing Architecture Advisory Service) Executive Report, Vol. 2, No. 4 (April 1999). ISSN 1523-5912.

[18] H.Kilov, H. Mogill, I. Simmonds. Invariants in the Trenches. *Object-Oriented Behavioral Specifications* (Ed. by H. Kilov and W. Harvey). Kluwer Academic Publishers, 1996, pp. 77-100.

[19] H.Kilov, J.Ross. *Information modeling.* Prentice-Hall, 1994.

[20] H.Kilov, K.P.Tyson. *Semantics Working Group Green Paper One.* OMG Semantics Working Group. OMG Document Number ormsc/97-06-10r.

[21] Yuri M. Lotman. *Universe of the mind: a semiotic theory of culture.* Translated by Ann Shukman. Introduction by Umberto Eco. London and New York: I.B.Tauris & Co. 1990.

[22] Herbert W. Lovelace. The medium is more than the message. *Information Week*, July 15, 2001.

[23] http://informatics.mayo.edu/index.php?page=11

[24] K.Riemer. An analysis of RM-ODP viewpoints and system life cycles. In: *Proceedings of the 8th OOPSLA workshop on behavioral semantics* (Ed. by K.Baclawski, H.Kilov, A.Thalassinidis, K.Tyson). Northeastern University, 1999.

[25] RM-ODP-2. ISO/IEC JTC1/SC21. Open Distributed Processing - Reference Model: Part 2: Foundations (ITU-T Recommendation X.902 | ISO/IEC 10746-2).

[26] RM-ODP 3. ISO/IEC JTC1/SC21. Open Distributed Processing - Reference Model: Part 3: Architecture (ITU-T Recommendation X.903 | ISO/IEC 10746-3).

[27] *Requirenautics Quarte*rly (The Newsletter of the Requirements Engineering Specialist Group of the British Computer Society), Issue 28 (February 2003). .Ian F.Alexander. Using formal methods to understand requirements better., 4-6.

[28] Ian Sommervile. MDA revisited (Letter to the Editor). *IEEE Software*, July/August 2004, pp. 9-10.

[29] Software Engineering. Report on a Conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968. (Chairman: Professor Dr. F.L. Bauer, Co-chairmen: Professor L. Bolliet, Dr. H.J. Helms; Editors: Peter Naur and Brian Randell). January 1969.

[30] *Software Engineering Techniques*. Report on a Conference sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969. (Chairman: Professor P. Ercoli, Co-Chairman: Professor Dr. F.L. Bauer, Editors: J.N. Buxton and B. Randell) April 1970.

[31] Lawrence E. Sweeney, Enrique V. Kortright and Robert J. Buckley. Developing an RM-ODP-based architecture for the Defense Integrated Military Human Resource System. *Proceedings of the WOODPECKER-2001 (Open Distributed Processing: Enterprise, Computation, Knowledge, Engineering and Realisation) in conjunction with ICEIS'2001*, Setubal, Portugal, July 2001, pp. 110-123.

[32] [Semantic web] Hewlett-Packard. Introduction to Semantic Web technologies. http://www.hpl.hp.com/semweb/sw-technology.htm

[33] W.Turski. It was fun. *Information Processing Letters*, 88(2003), 7-12.

[34] A.Thalassinidis and I.Sack. Building the Industry Library — Pharmaceutical. In: *Second ECOOP Workshop on Precise Behavioral Semantics (with an Emphasis on OO Business Specifications).* (Eds. H. Kilov and B. Rumpe), Technische Universität München, TUM-I19813, June 1998, pp. 245-253.

[35] Gerald Weinberg. *Rethinking systems analysis and design.* Little, Brown, and Company, 1982.

[36] Jeannette M. Wing. Hints to specifiers. In: *Teaching and learning formal methods.* (Eds. C. Neville Dean and Michael G. Hinchey). Academic Press, 1996, pp.57-77.

[37] Yair Wand, Ron Weber. Reflection: Ontology in information systems. *Journal of Database Management*, 15, 2 (April-June 2004), iii-vi.

# What Foundations does the RM-ODP Need?

Peter F. Linington.

*University of Kent,*
*Canterbury, Kent, CT2 7NF, UK.*
*pfl@kent.ac.uk*

## Abstract

*This position paper revisits the requirements for the set of Foundation Concepts for the ODP Reference Model and the approach originally taken to satisfying them. It then examines, in the light of experience, the areas where the Foundations have subsided, and areas where extensions need to be built. The aim is to provide a starting point for discussion on requirements to change the Foundations document.*

## 1. Introduction

Even before the first draft of the Reference Model for ODP was produced, the group of experts working on it had found the need for a separate definition of a clear conceptual framework on which to base their work; almost ten years later, this became Part 2 of the published standard [1], and established the conceptual framework for the ODP Architecture [2] [3]. The need for the RM-ODP Foundations document was clear; experts from a number of different backgrounds had come together to work on ODP, and they brought with them a wide range of different vocabulary and usage, reflecting different assumptions about how systems should be structured and specified. Progress with a common reference model depended on the creation of a common conceptual framework.

At the same time, no single notation or descriptive technique could be expected to dominate. The broad scope of the work was such that different techniques would be needed to express different areas of concern. It was therefore necessary to express the Foundations in abstract terms, bringing together the common features of existing styles of usage, so that each concrete notation is seen as a refinement of the foundation concepts. This is particularly the case in the areas of interaction and behaviour, which are discussed below.

In reviewing how well the RM-ODP has stood the test of time, we must be aware of the constraints on modifying it. Not only must the Parts of the reference model maintain their internal consistency, but the documents that reference it must not be undermined. This applies both to the ISO standards within the framework the reference model has created, and to the work within bodies such as the ITU-T and the OMG, and also the usage within the wider community. We do not have a green field; we have a responsibility to perform restoration, not demolition and replacement. The Foundations may need to be strengthened, but not relaid.

## 2. Objects and Interactions

At the time that the Foundations was being debated, two formal description techniques were also being developed within the same parent standards committee. These were LOTOS [4] and ESTELLE [5]; they differ significantly in their representation of interaction and this coloured the discussions in ODP. In LOTOS, the processes interact at gates in a synchronous way, in that all the parties to an interaction would agree on when the interaction occurs (although basic LOTOS deals with action sequence, not timing). In ESTELLE, on the other hand, modules are linked by communication links, which contain umbounded queues, so that interaction is represented by distinct sending and receiving actions that are sequenced but not simultaneous.

This led to a need for a common modelling basis that was capable of unifying both approaches. Interactions in ODP are synchronous, but are defined between an object and its environment. If we then constrain the way objects are composed so that each object binds directly to another object in its environment, the resulting communication is synchronous, but if objects bind to link ends in their environment, the resulting communication is asynchronous.

One might feel that the asynchronous representation was slightly cumbersome, but the Foundations also provide the less closely coupled concept of communication, which represents a sequence of causally related interactions, so that channels between communicating objects can (but need not) be completely hidden in this style of notation.

Currently, the Foundations do not distinguish between different kinds of interaction, but leave it to the specification using them to refine the basic concept. This, together with the connotations in English of the work "interaction", may have given the impression that the intension is something like a method call. This could be avoided and the full generality demonstrated by including in the standard non-exclusive definitions for some common refinements of interaction, such as invocation, message transfer and event notification.[1] For brevity, these examples are taken from a computational domain, but the definitions added should be viewpoint-independent, with notes to clarify their application, including examples in at least the engineering and enterprise domains.

## 3. Interface

The Foundations define an interface in terms of a view of the behaviour of an object, resulting from taking a subset of the interactions of the object and hiding all the other interactions and behavioural constraints that involve them. This definition is basically sound, but there are some subtleties that need to be taken into account in order to understand it fully.

Let us consider a computational application of the definition to see some of the problems; assume a style of interaction in which there is a clear causal initiative, so that an object is invoked by its clients and may, as a result, invoke other objects providing services (see figure 1).

The object, Obj, has four interfaces and its complete behaviour places constraints within and between them. Its environment contains four objects, A to D. The figure shows that:

a) object A can initiate interaction P at interface $If_1$ at any time;

b) object B can initiate interaction Q at interface $If_2$ at any time;

c) object B can initiate interaction R at interface $If_2$ but constraint $C_1$ requires Q to happen first; when R does occur, it is followed by Obj initiating interaction T with object D at $If_4$;

d) object B can initiate interaction S at interface $If_2$ at any time; when S does occur, it is followed by Obj initiating interaction U with object C at $If_3$; the behaviour of C results in an interaction V with Obj at interface $If_2$.

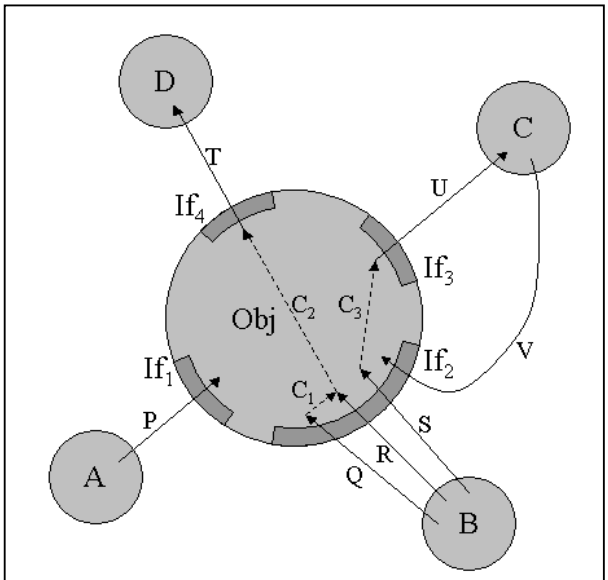



**Figure 1 – Interfaces and Object Behaviour**

Now let us see how the definition of the interface concept works. The first thing to note is that all the interactions of Obj are considered, and not just those in which it acts as a responder. Thus we are dealing with four interfaces, whereas a computational middleware might only consider interfaces 1 and 2 at which services are offered. Secondly, the interactions here are action occurrences, and not action types; actions P and Q could be of the same type, but the occurrences in the two interfaces are distinct. In fact, as a result of this, the sets of interactions are normally infinite, because most objects will have a behaviour that allows arbitrary repetition of smaller fragments of behaviour associated with some sort of thread or session; almost all notations simplify this by considering as a unit the sets of interactions with similar types and names (where the naming domain is associated with the interface). Note that this idea of equivalence sets implies that the identification of interfaces is a design activity – the interfaces cannot be deduced by examination of the overall computational behaviour (although, once the decision has been made, it is generally reflected by the naming structure used for interactions in the engineering viewpoint).

[1] The foundations originally avoided the use of the term event because of the connotations arising from the world of discrete event simulation; however, the use of the term event notification seems to avoid this.

Now let us consider the individual interfaces in the example. Interface 1 consists of occurrence of members of the set P without further constraint.

Interface 2 consists of the interaction sets Q, R, S and V, but subject to the constraint that an R must be preceded by an occurrence of Q. The constraints 2 and 3 are hidden in interface 2 because they involve interactions T and U, which are themselves hidden. The multi-step constraint between S and V is hidden for the same reason – it depends on U and also on the behaviour of C, which are themselves both hidden. These constraints only become apparent when the full behaviour of the object is considered; the link from S to V depends on specific behaviour in the environment, and so cannot be deduced from constraint 3 alone. However, a constraint between S and V could have been stated in interface 3 in terms of locally available properties, such as the presence of a correlation identifier as a data item in both the interactions.[2]

Interfaces 3 and 4 are again straightforward; apart from the placement of the initiative for interaction, they are structurally similar to interface 1.

What we have not yet considered is the dynamic lifecycle of an interface. Before interaction can take place, there are normally two steps to be taken. First, the object must be in a state where it is willing to interact (corresponding to the creation of the interface and its associated naming domain for interactions and initialisation of related internal state of the object) and second any interactions with the environment needed to establish preconditions for interactions at the interface must have been performed (corresponding to the establishment of a binding and associated communication and resultant state shared between the objects involved). After the first step, there is potential for interaction, but no specific partner for interaction has been selected, whilst after the second step interaction with specific partners can take place. Different kinds of object behaviour allow the description of one-to-one bindings or many-to-one client-server bindings. Similar considerations apply to the deletion of a binding and an interface.

When the Foundations were being drafted, a number of ways of modelling this process were considered, mostly based on including an intermediate concept for the unbound state, such as semi-interfaces or unbound interfaces, but none of them were successful. It would be worth revisiting this issue, but a better solution might be to consider a more general way of describing the potential behaviour of an object that is currently in a particular state (see section 8 below).

[2] This has different semantics, particularly with respect to the trust implications for C, but it may be equivalent for practical purposes.

## 4. Components

One of the significant changed in the last ten years has been the growth of interest in component-oriented architectures, so a natural question to ask is whether the Foundations should include a general definition of what a component is. If we consider, for example the CORBA Components 3.0 Specification (figure 2), and ask what the key properties of a component are, we find

a) encapsulation;

b) interactions at ports; the ports can be specialised as facets, receptacles, event sources, event sinks or attributes;

c) a component equivalent interface that provides metadata, navigation and control for the component;

d) an associated component home interface, representing a container in which components of the given type can be instantiated.

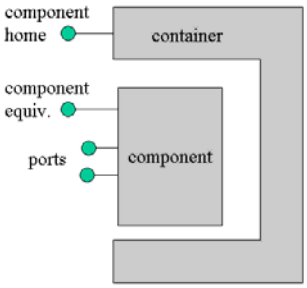



**Figure 2 – A component model**

In fact, all of these can be modelled using the existing Foundations. The general concept of interaction is rich enough to be refined into any of the defined port types, and the equivalent and home interfaces are just conventional computational interfaces. The container property requires a specialisation of the object concept to make explicit the relation to the instantiation of templates involved when it acts as a factory.

The ability to support both facets (server interfaces) and receptacles (client interfaces) as specialisations of the ODP concept of interface has already been mentioned above; it is primarily this generality, and the equivalent capabilities for interaction support, that make the foundation concept of object so flexible.

Perhaps a little more needs to be said about encapsulation. The Foundations explain encapsulation as the property of an object such that it can only have its state changed by interactions or internal actions expressed in the model. As such, the definition is more related to

completeness of description than to level of abstraction. Indeed, the inability to guarantee encapsulation on structural refinement is one of the problems in security analysis (see figure 3), for example, since structural refinements may introduce backdoors, and it is difficult to apply constraints to the refinement process that prevent this.

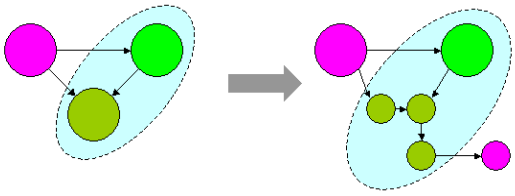


**Figure 3 – Refinement breaks a security boundary**

However, there would be nothing to choose between an object model and a component model built on the Foundations in this respect, even if our common intuition is that encapsulation of a component is a less abstract claim than encapsulation of an object.

As a result there is no need for extension of the Foundations if it was felt that there was a need to incorporate a computational or engineering component model into the architecture. However, there could be merit in adding derived definitions for component and factory, making it clear how the existing definitions can be used to model them.

## 5. Roles

The Foundations define a role as "an identifier for a behaviour, which may appear as a parameter in a template for a composite object, and which is associated with one of the component objects of the composite object". This definition has been much misunderstood, and some authors have tried to rework the definition in terms of static class structures, without much success. To attempt to do so is to miss the point of the definition, but clearly it does not, in its present form, convey the intent, which is indicated by the reference to templates and to the actualisation of parameters in its second paragraph (which is not quoted here).

The discussion here is based on the explanation in [6]. The metaphor on which the role concept is based is theatrical. The text of a play is expressed in terms of lines and actions associated with various roles, which are declared initially in a cast-list. Putting the play on involves assigning actors to the various roles, although one actor may play several minor roles, and the actor playing a role may change during the run of the production. Identifying the roles rather than the actors obviously makes the script more reusable.

The key idea is that some constraints on system behaviour are associated with objects dynamically as a consequence of an earlier part of the behaviour, such as performance of a piece of negotiation. However, although the potential behaviour can be referenced (and hence the talk of an identifier in the definition) it is not associable with an actual object until the template is instantiated and the role bound to a specific object[3]. It is thus impossible to represent what is going on within a static class hierarchy.

The solution to the misunderstandings is to remove the idea of there being a parameter identifier in the template's behaviour to the explanatory note, and to focus the first paragraph of the definition on the idea of parameter substitution. Perhaps more importantly, though, we need to clarify the way the potential behaviour of an object is restricted when it is bound to a role, and this needs a proper framework for the discussion of potential behaviour of the kind described in section 8 below.

The second problem with the role concept at present is with usage rather than definition. There has been widespread discussion in ODP circles of community-roles in the Enterprise Language, but unfortunately this has been expressed the term using role without qualification, leading to an implicit assumption that saying role implies community-role. As shown above, role is defined as a parameterization mechanism for templates, and so can potentially be applied to anything for which a template can be defined. Indeed, there are other places where the role concept is not just useful but is vital to making necessary distinctions in the template definition.

Perhaps the clearest present need is in the definition of action templates, particularly interaction templates. In an interaction between, say, a client and a server object, it is essential to know which is which. We can do this by saying that the two objects in this example fill client and server roles in the interaction, and by associating necessary properties and constraints with these roles. At one point in a system's behaviour, an object A can fill the buyer role in a purchase interaction, while object B fills the seller role, and later the roles can be reversed, so that B is the buyer and A is the seller. The richer the interaction, the more useful this

[3] Depending on the nature of the template involved, some roles may be bound after the instantiation of the object defining them. This is particularly true of objects representing potentially long-lived structures like communities, where the behaviour will commonly include the dynamics of community-role bindings (e.g. changes of committee membership). However, the lifetime of the role binding is always within the lifetime of the defining object, so that the objects created by a factory are not simply filling roles in it.

approach is likely to be; it is particularly effective, for example, for expressing the capabilities and obligations associated with secure multi-way interactions, where capabilities or access permissions are clearly associated with a specific role. Another example is for distinguishing between actor and artifact roles in enterprise interactions. Users of the role concept should be encouraged always to qualify their use of role with the template type name, as in action-role and community-role.

## 6. Obligations

The Foundations define a number of deontic concepts, particularly obligations, permissions and prohibitions, but this part of the framework was produced before there had been much experience with their application in ODP. The result is that the definitions were taken directly from the Standard Deontic Logic (SDL), including a simple set of relations between the concepts, such as the assertion that a permission for something is an indication that there is not an obligation not to do it.

There is nothing wrong with these as statements from the SDL, but experience has shown that the SDL approach is somewhat brittle for enterprise modelling, and it would be better to take a less prescriptive approach in the Foundations, allowing, for example, a style of modelling based on Utilitarianism to be exploited if it proves effective. See [8] for a discussion of how obligations might be represented in this way.

## 7. The lifecycle of ODP specifications

It has always been a principle in the development of ODP that the reference model is neutral with regard to the methodologies to be applied, and maintaining this position gives the most broadly applicable framework.

However, It is clear that most systems will evolve over time, and the reference model needs to take this into account. The Foundations includes the concept of an epoch to describe the way in which objects or configurations of objects evolve through a series of stages. This concept can also be used to describe the evolution of the specification itself. This allows a new version of a specification to describe, for example, how it might be introduced as a staged transition from the previous version.

One area where there must be an expectation of evolution and statements of constraints on it is in the definition of policies. The current Foundations definition of a policy is very weak. It is just defined as “a set of rules related to a particular purpose”, with an indication that the rules are expected to be expressed in deontic terms. There has been a great deal of work on the use of policies, particularly in various styles of policy-based management, since the creation of the reference model, and the requirements are now much better understood. We can now identify at least two stronger requirements for a rule to be considered a policy.

The first of these is that there must be some element of choice associated with any policy. Policies are identified in a specification wherever it is recognised that a rule may need to be changed during the lifetime of the specification; selecting a structure for the specification that emphasises the scope of the policy makes it easier to modify it without wholesale revision, and allows the likelihood of change to be reflected in the implementation. This is generally done by encapsulating associated decisions as the behaviour of a distinct computational or engineering object that can be replaced whenever the policy is changed.

Thus a rule that is universally true, and cannot be changed without wholesale replacement of the specification, is not a policy. The speed of light is not a policy, but the setting of interest on credit at a certain percentage above base rate is. Whether an organization operates with the status of a charity either might or might not be a policy, depending on whether the specifiers foresaw the possibility that the status might change and planned for it.

The second thing to be said about policy is that the specifiers who identify it will generally wish to limit the range of behaviour that would be acceptable; this gives rise to the idea of a policy envelope [7], which limits the range of behaviour any particular policy is allowed to specify. Knowledge of the policy envelope allows the verification of invariants on the specification that are independent of the particular policy in use at any particular instant.

Another aspect of the lifecycle of an ODP specification is the relationship between specification and instantiation, which is discussed in [6]. This paper identifies the need to enhance the ODP conformance model slightly so that it is able to distinguish between classes of use to which the specification is being put. The current ODP conformance model describes the relations between system the specifier, the implementor and the tester, and describes how conformance is deduced from observation during testing, confirming that the implementation is consistent with the original specification. A proposal made in [6] is to introduce the role of system owner, so that statements of rights to implement and use the design embodied in the specification can be made and then interpreted during the testing process to guide the interpretations made by the tester.

Making this comparatively minor extension opens the way to a more formal model of licensing and rights to use the design, and could help clarify the constraints on system

evolution involving reuse of design libraries or components.

## 8. Frames and unified semantics

Several of the areas examined so far have involved the need to describe the possibility of change or of the system responding to reaching a certain state of affairs or set of objectives. Constraints of the same basic kind are involved in basic behaviour like the interface and binding lifecycle, the definition of policies and the use or reuse of specifications. It would be a great aid to consistency of modelling if all these were based on a similar underlying model structure.

Most of the notations of practical interest here have their semantics defined denotationally, in terms of a mapping from notational elements to some mathematical target domain. For example, LOTOS is defined in terms of a labelled transition system. Other notations, such as UML, still lack a uniform and consistent semantic mapping. What is needed is a common target domain that is a natural extension of those in common use but with the power to be a target all the ODP-related notations, including the deontic aspects of enterprise languages.

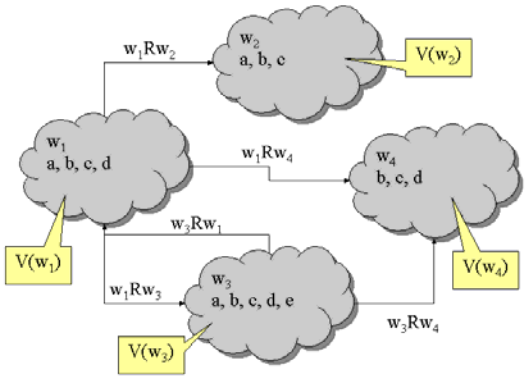



**Figure 4 – A frame-based model**

One possible direction would be to introduce a frame-based model such as one based on Kripke Frames (see [9] for an accessible review of these and other related systems able to support modal logics). The idea (see figure 4) is that the development or evolution of the system is represented by a model consisting of:

a) a set, W, of possible worlds of interest; the form of description of the world is not of interest here, except that it is decomposable into a set of attributes that carry Boolean values;

b) a dyadic relation, R, representing the reachability between members of W; R is true if $w_2$ is accessible from $w_1$, and false otherwise;

c) a value assignments, V, that assigns truth values to the attributes of each world in W.

This model captures in a mathematical way the intuition that we can describe any situation of interest for ODP as a set of worlds with local states of affairs and a set of statements as to whether any world can evolve into any other. Something is possible in a world if there exist worlds reachable from it in which that thing would be true, and something is necessary in a world if that thing would be true in all worlds reachable from it. Clearly, this kind of structure is capably of defining concepts like obligation, which is the deontic version of necessity.

From such a model we can go a step further by dividing the model into a set of related frames <W, R> and a separate set of value assignments, or markings, V. This is not quite as flexible as the general related worlds model, but does separate the structural aspects of evolution from the constraints on variables, making it easier to reason about. Frames of this kind are called Kripke Frames.

It would be possible to add such a frame definition to the clauses of the Foundations on basic interpretation concepts (putting mathematical detail in an annex, if necessary, to avoid possible intimidation of some readers) and thereby establish a common framework for supporting the basic behavioural and deontic aspects, and possibly the broader conformance and evolutionary issues as well.

Let us consider some of the problems discussed earlier in this light. First, consider the lifecycle of interfaces and bindings. The ability of an object to be involved in a particular kind of action or interaction can be represented as a marking in any particular world, and the ability to create an interface or perform a binding interaction is a special case of this; the actual performance of the interaction would result in changes to the markings between a world and any of its successors that are reached by performing the interaction.

Now, the existence of an interface can be deduced in a particular world from the existence of paths of succession between that world and worlds that are marked as allowing the associated interactions without, in so doing, traversing any successor step corresponding to interface creation – that is, if there is no creation between the point considered and some point of use. Essentially, what this is saying is that the property describing the existence of an interface is shared by all the worlds that can reach a world in which an interaction at that interface takes place (interaction is possible) without passing between a pair of worlds whose linking relationship corresponds to the creation of the interface. Although this may sound trivial, it gives a basis

for determining precisely the circumstances under which an interface exists, and so interactions at it can happen. It also implies some consistency conditions, in that, if two worlds are reachable by multiple paths, either all of the paths must include a step representing interface creation or none of them can.

Similar conditions can be applied to the existence of bindings. In this way we can describe the way in which objects are characterised by their potential to engage in bindings or specific interactions with an unbounded set of candidate peer objects, without requiring detail of the behaviour involved in their doing so. Clearly, the set of worlds that are equivalent in having the property that a binding exists must be a subset of the possible worlds that are equivalent in having the property that all the interfaces to be bound exist. However there is not a subset relationship between the sets in which each of the individual interfaces exist.

In a similar approach, we can capture the way in which the potential behaviour of an object is modified by creation of a community and by the object filling a community-role; filling the role results in a modification of the set of accessible worlds. The performance of an action commits the objects involved to playing their action-roles, and the preconditions for so doing can be derived from the accessibility relations.

Before leaving this issue, it should be stressed that what is being discussed here is a sketch of the definition of semantics for the basic modelling and specification concepts. There is no intention that the average user of these concepts would be involved in such considerations, but the unified underpinnings would give the basis for reasoning about the consistency of the framework and the correctness of interpretations by tools in difficult or potentially ambiguous cases.

## 9. Gaps and omissions

Finally, there are a number of areas in which the current Foundations standard omits material on the grounds that it is self evident or sufficiently obvious to be taken as read. Experience has shown that it is worth making even apparently well-understood concepts explicit if they are to be used in a formal way.

One example is the omission of terms in common technical usage, such as *relationship* or *association*, definitions of which should be included on the basis of significant usage in ODP specification, even if they seem obvious. The Foundations should include generic definitions consistent with, but less detailed and restrictive than those in the ISO General Relationship Model [10]. These generic definitions would then need to be related to the specific relationships that are already defined in the Foundations, such as the *subtype* and *subclass* relationships.

Another example is the omission of ODP specific detail or logical consequences of the existing definitions. Here one might consider the explicit definition of the concept of an *inter-viewpoint correspondence*, which is not discussed when *viewpoint* is defined. It should, indeed, be obvious to everyone that a system is only defined if both the viewpoints and the correspondences between them are detailed to a sufficient level to unify the overall specification, but sets of viewpoint specifications are often published without clear statements of correspondences, and a more balanced set of definitions would help just a little to get this message across.

Finally, the clause on specification concepts should be reviewed to check whether residual restrictions on usage are necessary. Many of the concepts, such as type and class, can be applied to a wide range of basic concepts. Thus, for example, the concept *type (of an <X>)* can be applied to any <X>. An individual specification can then declare which kinds of terms in it can have types. At present, however, composition only applies to objects, and refinement only applies to specifications. Discussion will be needed to determine whether these two concepts should be *of an <X>* or applied indirectly as relating specification fragments; either way, there would be a minor incompatible change to one or other of the concepts.

## 10. Conclusions

It seems that, in general terms, the ODP Foundations document has stood the test of time quite well. There is a need for some clarification of the definitions of roles and policies, and for addition of clear definitions of a number of concepts originally assumed to be well known, such as relation.

The most pressing need is for more explicit definitions relating to evolution in time, relating both to system behaviour (for example the lifecycle of interfaces) and, more generally, of a set of ODP specifications to reflect changes of requirement and policy. This can best be done by having an explicit representation of the possible worlds a specification applies to; this would need to be referenced as part of the most fundamental support for modelling used to define the basic and specification concepts.

Although the concept definitions are quite stable, they are not always used to best advantage in related standards. In particular, because the ODP Architecture was developed in parallel with the Foundations, there are places where usage in Part 3 is incorrect or where rules could be expressed more precisely by making best use of the foundation concepts. Part 1 could be improved by making

the usage of the concepts from the Foundations more consistent and complete. It would also be possible to improve Part 4 by interpreting some of the concepts directly in terms of the semantic domain for the formal description techniques rather than writing them in the techniques directly. However, it seems unlikely that there is enough expert effort available to attempt so major a task on Part 4, and the refinement of Parts 2 and 3 of the reference model should be the primary target at present.

## References


[1] ISO\IEC IS 10746-2, *Open Distributed Processing – Reference Model: Foundations*, 1996.

[2] ISO\IEC IS 10746-3, *Open Distributed Processing – Reference Model: Architecture*, 1996.

[3] P. F. Linington, *RM-ODP: The Architecture*, In K. Raymond and E. Armstrong, editors, *Open Distributed Processing: Experience with Distributed Environments*, pages 15-33. IFIP, Chapman and Hall, February 1995.

[4] ISO\IEC IS 8807, *Information processing systems -- Open Systems Interconnection – LOTOS: A formal description technique based on the temporal ordering of observational behaviour*, 1989.

[5] ISO/IEC IS 9074, *Information technology -- Open Systems Interconnection -- Estelle: A formal description technique based on an extended state transition model*, 1997.

[6] P. F. Linington and W. F. Frank, Specification and implementation in ODP, In J. Cordeiro and H. Kilov, editors, *Proceedings of the 1st Workshop on Open Distributed Processing: Enterprise, Computation, Knowledge, Engineering and Realisation*, pages 69-80, Setubal, Portugal, July 2001. ICEIS Press.

[7] P. F. Linington and S. Neal, *Using policies in the checking of business to business contracts*, In H. Lutfiyya, J. Moffat, and F. Garcia, editors, *Fourth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 207-218, Lake Como, Italy, June 2003. IEEE Computer Society.

[8] P. F. Linington, Z. Milosevic and K. Raymond, *Policies in Communities: Extending the ODP Enterprise Viewpoint*, In *Proceedings of 2nd International Workshop on Enterprise Distributed Object Computing (EDOC98)*, San Diego, USA, November 1998.

[9] G.E.Hughes and M. J. Cresswell, *A Companion to Modal Logic*, Methuen, London, 1984

[10] ISO/IEC IS 10165-7, *Information Technology – Open Systems Interconnection – Structure of management information: General relationship model*, 1996.

# Action Templates and Causalities in the ODP Computational Viewpoint

Raúl Romero and Antonio Vallecillo
*Dpto. de Lenguajes y Ciencias de la Computación*
*University of Málaga, Spain*
*{jrromero,av}@lcc.uma.es*

## Abstract

*The RM-ODP is a reference model that provides a coordinating framework for Open Distributed Processing standards, and offers a well-defined and comprehensive set of concepts and functions for the specification of ODP systems. Some years after its release as International Standard, an ISO Study Group will evaluate the need for a revision of RM-ODP, a customary process for ISO standards. The goal is to make use of the experiences gained from the use of the RM-ODP framework during that period, in order to propose improvements or changes if required. In order to serve as an input to that Group, this paper raises two small issues that we have discovered when trying to formalize the ODP Computational Viewpoint: the need of an independent term for referring to the signature of an Action Template, and the way in which Causalities are currently defined and handled. A proposal for addressing these issues is presented for discussion.*

## 1. Introduction

The ISO and the ITU-T jointly developed a Reference Model for Open Distributed Processing (RM-ODP) [8, 11-14], which provides the coordination framework for ODP standards, and creates an infrastructure within which support of distribution, interworking and portability can be integrated. The goal of this joint standardization effort is to define a reference model to integrate a wide range of future ODP standards for distributed systems and maintain consistency among them.

RM-ODP provides five generic and complementary viewpoints of the system and its environment: *enterprise*, *information*, *computational*, *engineering* and *technology*. Each of them has its own specific viewpoint language, defining concepts and rules for specifying ODP systems from the corresponding viewpoint.

The *enterprise viewpoint* focuses on the purpose, scope and policies of an ODP system. The *information viewpoint* describes the semantics of information and of information processing. The ODP *computational viewpoint* describes the functionality of a system and its environment, in terms of a configuration of objects that interact at interfaces. The *engineering viewpoint* focuses on the mechanisms and functions required to support distributed interactions between objects in the system. Finally, the *technology viewpoint* focuses on the choice of technology for that ODP system.

After formalizing the enterprise and the information viewpoints concepts [3, 4] using the Maude language and system [5, 6, 7], we recently started working on the formalization of the computational viewpoint specifications [2], for which other formalization efforts also exist [1, 9, 10, 15]. Our work has allowed us to explore the basic concepts defined in ODP, in addition to those specific to the computational viewpoint. Furthermore, some case studies have been developed, and a metamodel for the Computational Viewpoint has been proposed in [2]. The metamodel describes the concepts used in a computational viewpoint specification and the relationships between them.

In general, we find that Parts 2 [12] and Part 3 [13] of the ODP Reference Model are two excellent standards, fully consistent, and solidly conceived and architected. However, the inherent complexity of some of the concepts and functions defined in these two standards, their (sometimes) cryptic definition, and the lack of examples and real applications for most of the concepts, may hinder their understandability for readers which are not familiar with such terms. Having said that, we also discovered that, once understood, the concepts provided by RM-ODP are really valuable for the specification of open and distributed systems, and that everything fits in the conceptual framework with a clock-maker precision.

However, we also discovered that the use of these standards might help uncovering some small details that cannot be easily detected otherwise. Thus, based on our experiences with the case studies and the definition of the metamodel, we observed two issues in the ODP computational viewpoint. First, the term *Action Template* seems to cover both the syntactic (i.e., signature) and semantic (i.e., behavioral) aspects of an action template. The problem is that there is no specific concept for

referring just to the signature of an action template, which may seem to be required in some situations, as we shall later see. To solve this issue we propose, roughly, to include the concept *Interaction Signature*, which will specify just the syntactic part of an Action Template.

The second issue has to do with the way in which the concept of *Causality* is used. The Standard allows specifying causalities at different granularity levels (object, interface signature, and action template), but in an asymmetric manner. We propose a homogeneous treatment of causalities for both interface signatures and action templates.

In this paper we will discuss these two issues in more detail, together with the corresponding proposals for addressing them. Our proposals try to serve as an input for the current ODP revision work, and for discussion purposes.

The structure of this document is as follows. First, Section 2 describes inconsistencies found when dealing with the concept of *Action Template*. Section 3 deals with the distinction between *causalities* contained at the object, interface, and action template levels. Finally, Section 5 draws some conclusions.

## 2. Action Templates

The problem we found with action templates is about the way in which this concept is used for defining operation, signal, and stream signatures. In particular, the problem appeared in the metamodel when trying to model the existing relation between interface signatures, interaction signatures, and action templates.

First, according to Part 2 [12–9.11], a *Template* is "*the specification of the common features of a collection of <X>s in sufficient detail that an <X> can be instantiated using it*". From this definition, we directly obtain that an *Action Template* can be defined as "*the specification of the common features of a collection of actions in sufficient detail that an action can be instantiated using it*."

Then, we looked at how Action Templates are used in Part 3 in the Computational Viewpoint, in which they play a very relevant role.

First, we see that Part 3 indicates [13 – 7.1.12] that "*an announcement signature is an action template*" (the underlined text is ours). Another reference appears when referring to interrogation signatures. Although we expected a similar definition, what we find in Part 3 is that "*an interrogation signature comprises an action template*" [13 – 7.1.12].

The concept *action template* appears again when defining stream interface signatures, which comprise a finite set of action templates.

So the first issue is whether signatures "are" action templates, or "comprise" action templates.

Furthermore, there is the issue of the way in which action templates are used in Part 3 for defining signatures. Commonly, signatures (of both interactions and interfaces) are considered to remain at the syntactic level, i.e., they are supposed to describe just the names and types of the actions and their parameters. Semantic information (e.g., behavior) is not usually covered by signatures. However, this does not seem to be consistent with the use of action templates for defining signatures, since action templates might also include behavioral specifications (cf. Part 2).

Certainly, this is also corroborated by Part 4 [14 – 4.4.2.12], which states that: "*It should be noted that the text in ITU-T Rec. X.902 | ISO/IEC 10746-2 treats an interface signature as a set of action templates associated with the interactions of an interface. Given that an action template is likely to include semantic information as well as syntactic. Common interpretations of interface signature deal primarily at the syntactic level, however.* […]".

We propose to solve these two issues by introducing a term that refers to the syntactic information specified by an action template, and that we have called *Interaction Signature*. This term can be used to define the signatures of announcements, interrogations, terminations, signals and flows (see Figure 1), that now are Interaction Signatures. This does not contradict the current standard text and, in fact, allows the separation of the syntactic and the semantic information specified by an action template.

Moreover, interface signatures (an abstract class that simply generalizes operation, signal and stream interface signatures) now comprise sets of interaction signatures, which seems to be more in line with the intent of the RM-ODP standard.

Finally, and as shown in the figure, the parameters of the action template are now associated to the syntactic part of such action template, that is, to its *Interaction Signature*, which also seems to be more natural than attaching them directly to the *Action Template*.

## 3. Causalities

Clause 13.3 of Part 2 states that "*the identification of causality allows the categorization of roles of interacting objects*". Furthermore, that clause provides "*a basic set of roles*" and specifies that a "*causality implies a constraint on each behaviour of the participating objects while they are interacting*".

Meanwhile, Clause 7.1 of Part 3 defines where the indication of the causality must be defined in each case, and for each element. For signal interfaces, their interface signatures "*comprise a set of finite action templates, one*

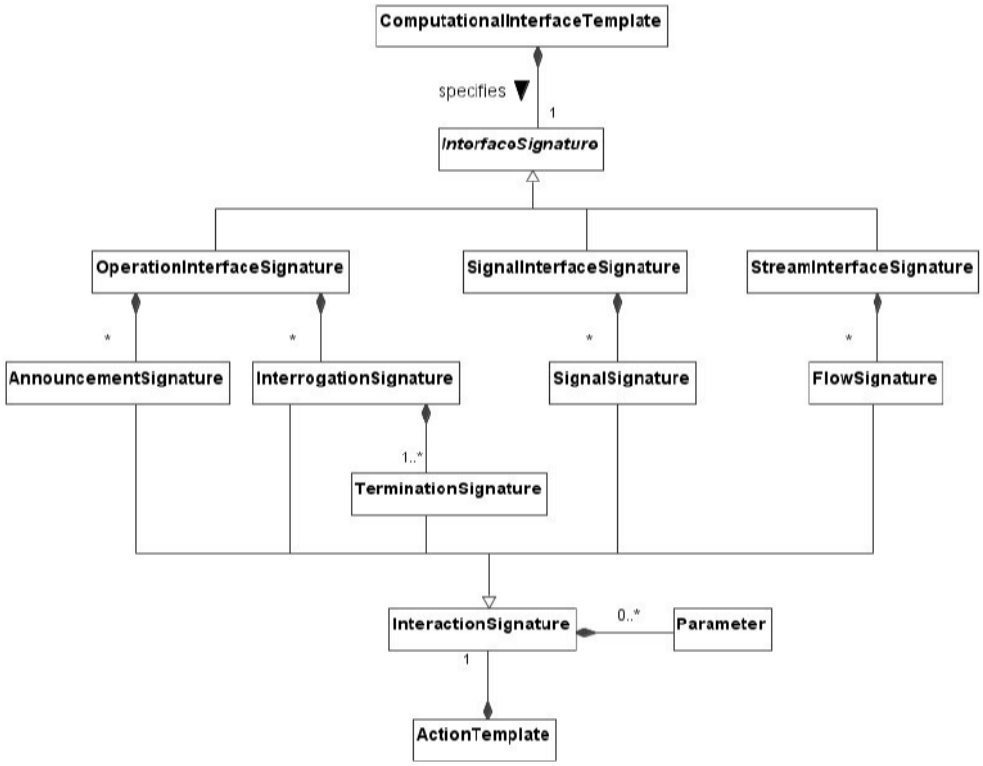



**Figure 1. Interaction signatures specify the syntactic information of Action Templates**

*for each type of signal in the interface. Each action template comprises the name of that signal, the number, name and types of its parameters and an indication of causality with respect to the object which instantiates the template*". Same for stream interfaces. However, for operation interfaces we noticed that causalities are not treated in the same way. In that case, the operation interface signature comprises, apart from a set of announcement and interrogation signatures, as appropriate, the indication of causality for the interface as a whole with respect to the object that instantiates the template.

Clause 7.2.2 of Part 3 (Interaction Rules) clearly refers to the causality "*in the interface's signature*", that seems to support the specification of causalities at the interface signature level. More precisely, sub-clauses 7.2.2.1 and 7.2.2.2 are clear and explicit when referring to this issue.

According to signal interaction rules [13 – 7.2.2.1], "*a computational object offering a signal interface of a given signal interface type*

- *initiates signals that have initiating causality in the interface's signature;*
- *responds to signal that have responding causality in the interface's signature.*"

Similarly, according to stream interaction rules [13 – 7.2.2.2], "*a computational object offering a stream interface*

- *generates flows that have producer causality in the interface's signature;*
- *receives flows that have consumer causality in the interface's signature."*

Thus, we find that, whereas the indication of causality for signal and stream interfaces is defined at the action template level, for operations it is defined at the object's interface signature level.

One of the reasons behind these decisions seems to be the fact that, for operations, the causality indication provided by the interface signature determines the causality for each action template in the interface, depending on what we are really using: invocations or terminations. However, when dealing with signal or stream interface signatures, which comprise signals or flows that may go in different directions (i.e., incoming and outgoing actions), there is no clear relationship between the causality of the interface signature, and the causalities of the individual interactions that comprise the interface signature. Thus, causalities should be defined

both for the interface with respect to the object that interacts and for each individual action template.

For example, let us consider a simple stream. Its signature could have both incoming and outgoing action templates defined for it. However, as mentioned in the interaction rules, we need to consider an indication of causality with respect to the role that the computational object plays in the communication process. This requires indicating that causality in the interface signature, which would indicate the object that produces the flow and the object that consumes it.

To address this issue, we propose to include causality definitions at both levels. However, calling it "causality" at both levels might be confusing too. Actually, the definition of causality in Part 2 refers to objects only, i.e., the granularity of causality is defined at the object level – more precisely at the object's interface signature level. But in Part 3, causality indications seem to be used at two different levels: object interface signature and action template. There is a clear need to align the granularities for these different definitions.

Thus, we propose defining causalities in every level at which this term is involved. This means incorporating causalities in individual action templates and in interface signatures. In operations, in which the causality defined at the interface signature level determines the causality of the individual interactions, a constraint should enforce such a relationship.

Figure 2 shows our proposal, where the indication of causality appears not only at the interface signature level—to specify the roles played by computational objects in the communication process as a whole—but also at the interaction signature level.

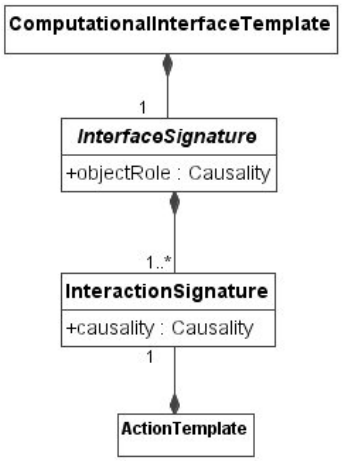



**Figure 2. Indication of causality at two levels**

## 4. Conclusions

RM-ODP was created at the beginning of last decade, but it is becoming now probably the best framework for specifying and developing large open and distributed applications. In the first place, the complexity of the applications is reaching the level where many traditional software engineering methods do not seem to be able to cope with. However, RM-ODP was specifically conceived to specify those large and complex open systems, and therefore is perfectly fit to address their specification and design. Furthermore, the level of maturity reached by the RM-ODP seems to be the adequate to fulfill the requirements of current businesses and organizations.

Some years after its release as International Standard, an ISO Study Group will evaluate the need for a revision of RM-ODP, a customary process for ISO standards. The goal is to make use of the experiences gained from the use of the RM-ODP during that period, in order to "tune" it according to the findings, and to propose improvements or changes if required.

In order to serve as an input to that Group, this paper has raised two issues that we discovered when trying to formalize the ODP Computational Viewpoint: the need of an independent term for referring to the signature of an Action Template (without taking into consideration the semantic information that an action template also contains), and the way in which Causalities are currently defined and handled. Proposals for addressing these issues have been presented. First, the term Interaction Signature has been proposed for capturing the syntactic aspects of an action template. This allows a consistent definition of all interaction signatures (announcements, interrogations, terminations, signals and flows), as shown in Figure 1. Second, we propose the definition of causalities at two levels: interface signature and interaction signature. This seems to resolve the apparent mismatch in Part 3 of the RM-ODP standard.

Finally, just to mention the need for more examples, case studies and documents describing experiences in the use of RM-ODP, in order to help software engineers fully understand the concepts in the Reference Model, whose complexity (and sometimes cryptic definitions) make them difficult to learn, understand, and properly use to specify and design large open distributed applications.

**Acknowledgements** The authors would like to acknowledge the work of many ODP experts who have been involved in investigating and addressing the problems of the computational specification of ODP systems. Although the views in this paper are the authors' solely responsibility, they could not have been formulated without the detailed discussions with ISO experts on ODP, in particular with Akira Tanaka and Dave Akehurst.

This work has been partially supported by Spanish Project TIC2002-04309-C02-02.

## 5. References

[1] D. H. Akehurst, J. Derrick and A.G. Waters. "Addressing Computational Viewpoint Design." In *Proceedings of the 7th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2003)*, pages 147-159, Brisbane, Australia, Sept. 2003. IEEE CS Press.

[2] R. Romero and A. Vallecillo. "Formalizing ODP Computational Specifications in Maude". In *Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004)*, Monterey, California, September 2004. Copyright IEEE CS Press.

[3] F. Durán and A. Vallecillo. Specifying the ODP information viewpoint using Maude. In H. Kilov and K. Baclawski, editors, *Proceedings of Tenth OOPSLA Workshop on Behavioural Semantics*, pages 44-57, Florida, Oct. 2001. Northeastern University.

[4] F. Durán and A. Vallecillo. Formalizing ODP Enterprise specifications in Maude. *Computer Standards & Interfaces*, 25(2):83-102, June 2003.

[5] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseger and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187-243. Aug. 2002.

[6] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer and C. Talcott, Maude 2.0 manual. Available in http://maude.cs.uiuc.edu, June 2003.

[7] S. Eker, J. Meseguer and A.Sridharanarayanan. The Maude LTL model checker. In F. Gaducci and U. Montanari, editors, *Proc. Of the 4th International Workshop on Rewriting Logic and its Applications (WRLA 2002)*, volume 71 of *Electronic Notes in Theoretical Computer Science*, pages 115-142, Pisa, Italy, Sept. 2002. Elsevier.

[8] P. Linington. *RM-ODP: The architecture*. In K. Milosevic and L. Armstrong, editors, Open Distributed Processing II, pages 15-33. Chapman & Hall, Feb. 1995.

[9] E. Najm and J.B. Stefani. A formal operational semantics for the ODP computational model. *Computer Networks and ISDN System*, 27:1305-1329, 1995.

[10] E. Najm and J.B.Stefani. Computational models for open distributed systems. In H. Bowman and J. Derrick, editors, *Proceedings of FMOODS'97*, pages 157-176, Canterbury, 1997, Chapman & Hall.

[11] ITU-T Recommendation X.901 | ISO/IEC 10746-1: *ODP Reference Model Part 1. Overview*. Geneva, Switzerland, 1998.

[12] ITU-T Recommendation X.902 | ISO/IEC 10746-2: *ODP Reference Model Part 2. Foundations*. Geneva, Switzerland, 1996.

[13] ITU-T Recommendation X.903 | ISO/IEC 10746-3: *ODP Reference Model Part 3. Architecture*. Geneva, Switzerland, 1996.

[14] ITU-T Recommendation X.904 | ISO/IEC 10746-4: *ODP Reference Model Part 4. Architectural semantics*. Geneva, Switzerland, 1998.

[15] R. Sinnot and K.J. Turner. Specifying ODP computational objects in Z. In E. Najm and J.B. Stefani, editors, *Proceedings of FMOODS'96*, pages 375-390, Canterbury, 1997. Chapman & Hall.

# The role of the RM-ODP Computational Viewpoint Concepts in the MDA approach[1]

João Paulo Almeida, Marten van Sinderen, Luís Ferreira Pires
*Centre for Telematics and Information Technology, University of Twente*
*PO Box 217, 7500 AE Enschede, The Netherlands*
*{almeida, sinderen, pires }@cs.utwente.nl*

**Abstract**

*An MDA design approach should be able to accommodate designs at different levels of platform-independence. We have proposed a design approach (in [2] and [3]), which allows these levels to be identified. An important feature of this approach is the notion of abstract platform. An abstract platform is determined by the platform characteristics that are relevant for applications at a certain level of platform-independence, and must be established by considering various design goals. In this paper, we define a framework that makes it possible to use RM-ODP concepts in our MDA design approach. This framework proposes a recursive application of the computational viewpoint at different levels of platform-independence. This is obtained by equating the RM-ODP notion of infrastructure to our notion of abstract platform.*

*Keywords*: Reference Model for Open Distributed Processing (RM-ODP), Model-Driven Architecture (MDA), platform-independence, abstract platform, design of distributed applications

## 1. Introduction

The Model-Driven Architecture (MDA) [13, 16] represents a prominent trend in the development of distributed applications. The concept of platform-independence plays a central role in MDA. A common pattern in MDA development is to define a platform-independent model (PIM) of a distributed application, and to apply (parameterised) transformations to this PIM to obtain one or more platform-specific models (PSMs). Significant benefits of this approach are that PIMs can be reused to target different technology platforms, and that PIMs are unlikely to be affected by platform evolution.

In our previous work [2], we have observed that the level of platform-independence at which PIMs are specified should be derived from application requirements and characteristics of the potential target platforms. In addition, in order to bridge the gap between requirements and implementation, it may be necessary to use models at different levels of platform-independence.

In [2, 3], we have proposed a design approach that introduces the concept of abstract platform. This concept supports a designer in identifying the level(s) of platform-independence at which PIMs are specified. An abstract platform defines an acceptable platform from an application developer's point of view; it represents the platform support that is assumed by the application developer at some point in (the platform-independent phase of) the design trajectory. Alternatively, an abstract platform defines characteristics that must have proper mappings onto a set of concrete target platforms, thereby defining a level of platform-independence. Defining an abstract platform forces a designer to address two conflicting goals: (i) to achieve platform-independence, and (ii) to reduce the size of the design space explored for platform-specific realization.

Any design approach that is intended to be successfully applied in practice should be supported by suitable design concepts. In this paper we define a framework that makes it possible to use RM-ODP concepts in our MDA design approach. This is obtained by equating the RM-ODP notion of infrastructure to our notion of abstract platform. This framework allows a recursive application of the Computational Viewpoint at different levels of platform-independence.

This paper is further structured as follows: section 2 reviews the notions of platform-independence and abstract platform as adopted in this paper, section 3 discusses the RM-ODP concepts that are of particular relevance to our work, section 4 applies these concepts in our MDA design trajectory, and section 5 discusses some related work. Finally, section 6 presents some conclusions and open issues.

## 2. Platform notions

Platform-independence [16] is a quality of a model that relates to the extent to which the model abstracts from the characteristics of particular technology platforms. For the purpose of this paper, we assume that platform corresponds ultimately to some specific middleware technology, such as CORBA/CCM [14, 15], .NET, or Web Services [21, 22], in which distributed applications are realized.

[1] This is a revised version of a paper that appeared in the Proceedings of the 1st European Workshop on Model-Driven Architecture with Emphasis on Industrial Applications (MDA-IA 2004), University of Twente, The Netherlands, March 2004.

Currently, a large number of middleware platforms are available (a small sample of these can be found in the latest proceedings of the ACM/USENIX Middleware conference [7]). Different middleware platforms provide different levels of support for applications. For example, there are platforms that offer confidentiality for distributed interactions, that implement transparent load-balancing mechanisms, or that provide some capabilities for dynamic upgrade of application components. Platforms may also differ in the interaction patterns they support, such as *request/response*, *message passing*, *message queues* and group communication mechanisms. As a consequence, the design of an application in terms of a particular middleware platform is *platform-specific*, since: (i) the design depends on particular technological conventions adopted by the middleware platform; (ii) the structure of the application depends on the set of interaction patterns supported by the platform; and (iii) the functionality addressed at application level depends on the services provided by the platform.

### 2.1. Levels of platform-independence

Model reusability with respect to platforms can be obtained by making these models platform-independent. Ideally, one could strive for PIMs that are absolutely neutral with respect to all different classes of middleware platforms. This is possible for models in which the characteristics of supporting infrastructure are irrelevant, such as, e.g., conceptual domain models [5] and ODP Enterprise Viewpoint models [10] (which can be considered Computation Independent Models [16] in MDA terms). However, along a development trajectory, when system architecture is captured, different sets of platform-independent modelling concepts may be used, each of which is adequate only with respect to specific classes of target middleware platforms. This leads to the observation that there can be several PIMs, including various levels of platform-independence, to be identified by a designer.

When different levels of platform-independence are necessary, they must be carefully identified. We propose to make this identification an explicit step in MDA development. The notion of abstract platform, as proposed initially in [2] and elaborated in [3], supports a designer in this step.

### 2.2. Abstract platform

An abstract platform is determined by the platform characteristics that are relevant for applications at a certain platform-independent level. For example, if a platform-independent design contains application parts that interact through operation invocations, then operation invocation is a characteristic of the abstract platform. Capabilities of a concrete platform are used during platform-specific realization to support this characteristic of the abstract platform. For example, if CORBA is selected as a target platform, this characteristic can be mapped onto CORBA operation invocations.

Characteristics of an abstract platform may be *implied* by the choice of design concepts used for describing the platform-independent model of a distributed application. These concepts are often directly related to the adopted modelling language. For example, the exchange of "signals" between "agents" in SDL [11] may be considered to define an abstract platform that supports reliable asynchronous message exchange. These concepts may also be specializations of concepts from the adopted modelling language. This can be the case with UML, which is specialized in order to suit the needs of platform-independent modelling, e.g., as specified in the EDOC UML Profile [18].

Instead of implying an abstract platform definition from the adopted set of design concepts for platform-independent modelling, it can be useful or even necessary to define some characteristics of an abstract platform *explicitly*, resulting in one or more separate and thus reusable design artefacts. During platform-independent modelling, a pre-defined abstract platform model may be composed with the model of the distributed application. For example, while UML 2.0 does not support group communication as a primitive design concept, it is possible to specify the behaviour of a group communication sub-system in UML. This sub-system can be re-used in the design of a distributed application that requires group communication. Other examples of pre-defined artefacts that may be included in abstract platforms are the ODP trader [9] and the OMG pervasive services (yet to be defined [16]).

We argue in the following sections that the RM-ODP Computational Viewpoint concepts are useful for specifying platform-independent designs. Our proposed framework makes use of both the implicit and the explicit approaches to define abstract platforms.

## 3. RM-ODP in application design

The ISO/ITU-T RM-ODP (Reference Model for Open Distributed Processing) [9] provides a specification framework for distributed systems development based on the concept of *viewpoints*. For each viewpoint, concepts and structuring rules are provided, defining a conceptual framework for specifications from that viewpoint. The use of different viewpoints in the design of complex systems is an accepted technique to achieve separation of concerns. This also has been reflected in standards such as, e.g., IEEE 1471 [8].

The RM-ODP computational and engineering viewpoints are relevant to the purpose of our work since they focus on application and infrastructure concerns, respectively.

### 3.1. Concepts in the computational viewpoint

The computational viewpoint is concerned with the decomposition of a distributed application into a set of interacting objects, abstracting from the supporting distribution infrastructure. In contrast, the engineering viewpoint focuses on the infrastructure required to support distributed applications. It is concerned with properties and mechanisms required to overcome problems related to distribution (e.g., remoteness, partial failures, heterogeneity) and to exploit distribution capabilities (e.g., to achieve performance and dependability), but that are abstracted from in computational viewpoint specifications.

The RM-ODP relies on the concept of (distribution) transparency, which is defined as the property of hiding from a particular user (or developer) the potential behaviour of some parts of a system [9]. In the context of the computational and engineering viewpoints, transparency is used to hide mechanisms that deal with some aspect of distribution. An example of distribution transparency is replication transparency, which hides the possible replication of an object at several locations in a distributed system. In the computational viewpoint, a single computational object would be represented, while this computational object may possibly correspond to several replica objects in the engineering viewpoint. The mechanisms necessary to ensure replica consistency and management are addressed in the engineering viewpoint, shielding the (computational viewpoint) designers from the burden of developing these mechanisms. Distribution transparency is selective in ODP; the Reference Model includes rules for selecting transparencies. Transparencies are constraints on the mapping from a computational specification to a specification that uses specific ODP functions and engineering structures to provide the required transparency.

In the computational viewpoint, applications consist of configurations of interacting *computational objects*. A computational object is a unit of distribution characterized by its behaviour. A computational object is encapsulated, i.e., any change in its state can only occur as a result of an internal action or as a result of an interaction with its environment. An object is said to have *interfaces*, each of which expose a subset of the interactions of that object. Interaction between objects is only possible if a *binding* can been established between interfaces of these objects. The computational viewpoint supports arbitrarily complex bindings, through the concept of binding object, which represents the binding itself as a computational object. The behaviour of a binding object determines the interaction semantics they support. As with any other object, binding objects can be qualified by quality of service assertions that constrain their behaviour. The computational model does not restrict the types of binding objects, allowing various possible communication structures between objects to be defined [9].

### 3.2. The RM-ODP notion of infrastructure

In [6], Blair and Stefani have equated the boundary between the computational and the engineering viewpoints to the distinction between application and infrastructure: “It is important to realize that the boundary between the two viewpoints is fluid, depending on the level of the virtual machine offered by the system’s infrastructure. Some systems will provide a rich and abstract set of engineering objects whereas others will provide a more minimal set of objects leaving more responsibility to the applications developer.” Specifications in the computational viewpoint are, according to this interpretation, influenced by the level of support provided by the infrastructure. By setting the level of support provided by the infrastructure, one can refer to computational concerns and engineering concerns.

Equating infrastructure to predefined middleware platforms would lead us to the conclusion that computational specifications are directly influenced by the level of support provided by a selected middleware platform. Computational specifications would therefore be, to some extent, platform-specific. In this case, the separation of computational and engineering concerns would be identical to the separation between application and middleware platform concerns. The reusability of a computational viewpoint specification would be restricted by its dependence on platform characteristics. Furthermore, from the perspective of application developers, the separation of computational and engineering concerns would be implied by the availability of a software infrastructure. Therefore, we conclude that the motivation for the separation of computational and engineering concerns is predominantly bottom-up.

Another interpretation for the infrastructure assumed by the computational viewpoint is that of an ‘ideal infrastructure’. In this interpretation, the motivation for the separation of computational and engineering concerns is predominantly based on the needs of the developer to handle the complexity of application and infrastructure separately, regardless of the availability of a software infrastructure. The engineering viewpoint offers the possibility for a designer to engineer the infrastructure explicitly. While this interpretation is ideal from the perspective of separation of concerns for the application developer, it does not leverage the reuse of middleware platforms, which would significantly improve the efficiency of the development process.

**Table 1 Interpretations of infrastructure compared**

| Interpretation (infrastructure equals to) | Reuse of middleware | Separation of concerns | Platform-independence |
|---|---|---|---|
| Available middleware platform | Yes | Based on target platform | Low |
| Required middleware platform (ideal from application point of view) | No explicit consideration | Defined by designer's needs; motivated by complexity in application design | High |

Table 1 summarizes the implications of these contrasting interpretations of infrastructure. We conclude that both interpretations considered have limitations when applied in conjunction with the MDA approach, which inspired us to investigate an alternative.

## 4. RM-ODP infrastructure notion revisited

Committing to one of the previously discussed interpretations of infrastructure is undesirable for the adoption of computational viewpoint concepts in the MDA. It may lead to models at a low level of platform-independence, or it may lead to models which cannot be realized on existing middleware platforms. We propose to equate the term infrastructure, as used in RM-ODP, to our notion of abstract platform. This approach can be beneficial for the development of distributed applications, so that a proper balance can be obtained between the following design goals:

- designers can use the separation of application and infrastructure concerns to cope with the complexity of distributed application design;
- middleware platforms can be reused to improve significantly the efficiency of distributed application development; and
- platform-independence can be obtained as a means to preserve investments in application development and withstand changes in technology.

A consequence of equating infrastructure to abstract platform is that computational viewpoint concepts can be applied recursively at different levels of platform-independence. The use of the same conceptual framework for different levels of platform-independence facilitates the definition of correctness relations or even automated transformations.

An abstract platform is defined in terms of the bindings supported, the transparencies supported, and the types of quality-of-service (QoS) constraints that may be applied to interface contracts. The use of binding objects may provide considerable flexibility to implementations of platform-independent models, since it is possible to provide countless different implementations of a binding object. In addition, there is considerable freedom in choosing mechanisms for obtaining a required transparency and satisfying QoS constraints.

At any point in a design trajectory, a mapping to a platform-specific realization may be defined, as long as: (i) the semantics for the original model is respected, as defined by the computational language; and (ii) quality characteristics of the realizations obtained through mappings are acceptable.

### 4.1. Example: simple conference application

In order to illustrate the use of computational viewpoint concepts along our model-driven design trajectory, let us consider a conference service that facilitates the interaction of users residing in different hosts. Initially, the service designer describes the service solely from its external perspective, as a conference binding object, revealing its interfaces and relating interactions that occur at these interfaces. Figure 1 shows a snapshot of the conference application with three user objects fulfilling the role of conference participant and a user object fulfilling the role of conference manager. Since characteristics of the internal structure of the binding object are not revealed, the user objects are specified at a high level of abstraction. The abstract platform at this level of abstraction supports the interaction between user objects and the conference binding object. The interfaces are described in terms of the ODP concepts of operation and signal.

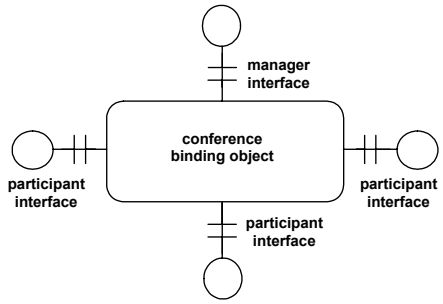



**Figure 1 Snapshot of conference application**

This example reveals the flexibility of the specification at this level of platform-independence. The conference binding object may be further decomposed into a centralized or distributed, symmetric or asymmetric design, and different abstract platforms may be used to support the interactions of the objects that implement it. Any number of recursive decompositions of the computational objects may be applied as necessary.

One possible way to proceed with design is shown in Figure 2. In this design, the internal structure of the conference binding object is revealed. The conference binding object is refined into a multicast binding and computational objects interconnected through this binding. The abstract platform at this level of abstraction supports multicast bindings as prescribed in the definition of the service of the multicast binding object.

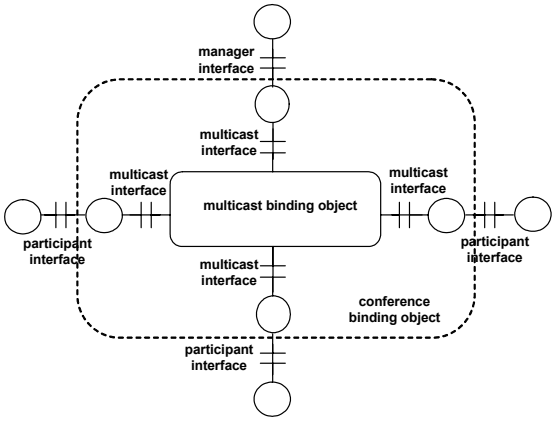



**Figure 2 Revealing binding decomposition**

At this point in the design trajectory, a mapping can be used to realize this design on top of a target platform that offers a multicast binding corresponding to that provided by the abstract platform. The engineering structures required to provide an adequate level of support are provided by the concrete platform. An alternative mapping could implement the multicast binding as a centralized object, realizing the interactions between the objects and the multicast binding object as distributed interactions. However, this alternative mapping may prove to be inadequate with respect to its quality-of-service characteristics, e.g., since a centralized implementation may fail to satisfy performance and scalability requirements. This flexibility in mapping is possible because the refinement of the conference binding in the computational viewpoint does not commit to a particular distribution in terms of nodes, capsules and clusters, as would have been the case with a refinement in the engineering viewpoint.

When the target platform does not provide the required level of support, the design can be further detailed in an abstract platform at a lower level of platform-independence. The refinement depicted in Figure 3 assumes an abstract platform that only supports binary bindings of operational interfaces. This mapping differs from the previous design steps in that it does not consist solely of decompositions.

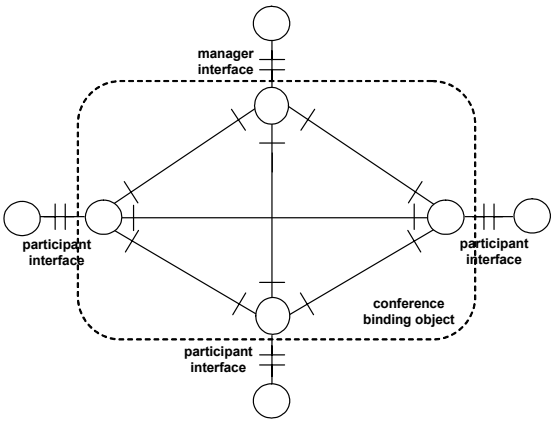



**Figure 3 Revealing binding decomposition**

The development trajectory that results from our approach as applied to the example above is illustrated schematically in Figure 4.

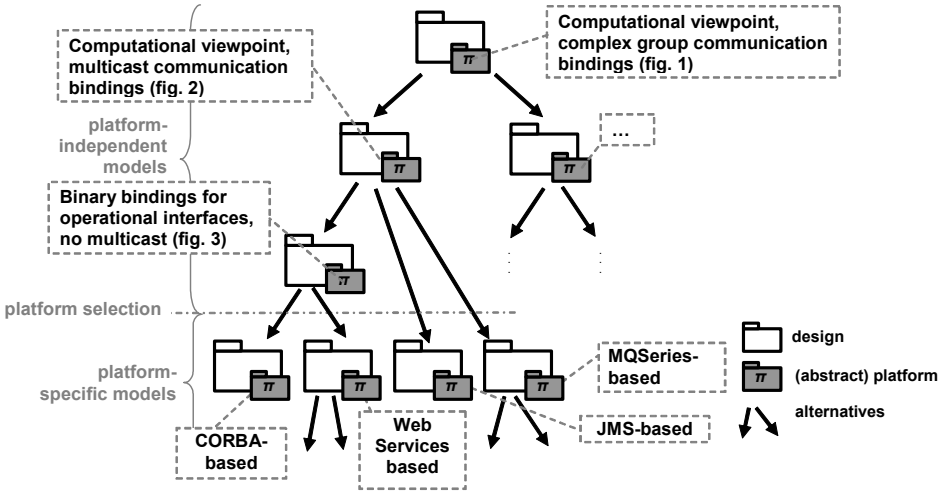



**Figure 4 Models at related levels of platform-independence**

### 4.2. Example: replication transparency

An example that reveals the role of transparencies in the design trajectory is presented in Figure 5. In this example, a client and a server object interact through an operation interface. A replication transparency schema is used to specify constraints on the availability and performance of the server object. Two different mappings of the source model (a) are depicted below. In Figure 5 (b), a realization is obtained by mapping the source model directly to a platform that supports replication transparency, namely, Fault Tolerant CORBA. The infrastructure depicted is provided with this platform [14]. In Figure 5(c), a realization is obtained by mapping the source model into a target model that explicitly addresses the replication of the server object. A replication object is introduced to execute the replication function, delegating requests to the different replicas. For simplicity, we consider stateless server objects, and therefore we can omit extra interfaces required for checkpointing. A possible realization of the application in Web Services [21, 22] is depicted schematically in Figure 5 (d).

The list of transparencies defined in the RM-ODP is not exhaustive. In [4] we have discussed the role of replacement transparency in an MDA design trajectory.

## 5. Related work

The ITU-T X.906 | ISO/IEC 19793 Working Draft [12] proposes the use UML profile for EDOC [18] to model the computational viewpoint. This profile provides the notion of recursive component collaboration which corresponds to the notion of computational object in the RM-ODP. However, no notion of selective transparencies is provided in the EDOC profile. Furthermore there is no support for the specification of QoS constraints. The EDOC profile may be considered to define a single implicit abstract platform: interactions in the EDOC profile are always decomposed into asynchronous interactions through "Flow Ports".

In [1], Akehurst et al. have focussed on the representation of the computational viewpoint concepts using MDA core technologies, namely UML and UML profiling. Putman [20] has also proposed some extensions to UML to accommodate the use of ODP design concepts. In this paper, we investigate the role of ODP concepts with respect to design goals introduced by the use of platform-independent models. Both references [1, 20] can be seen as complementary to the framework proposed in this paper, and the representations they propose may be applicable to the design trajectory we have discussed.

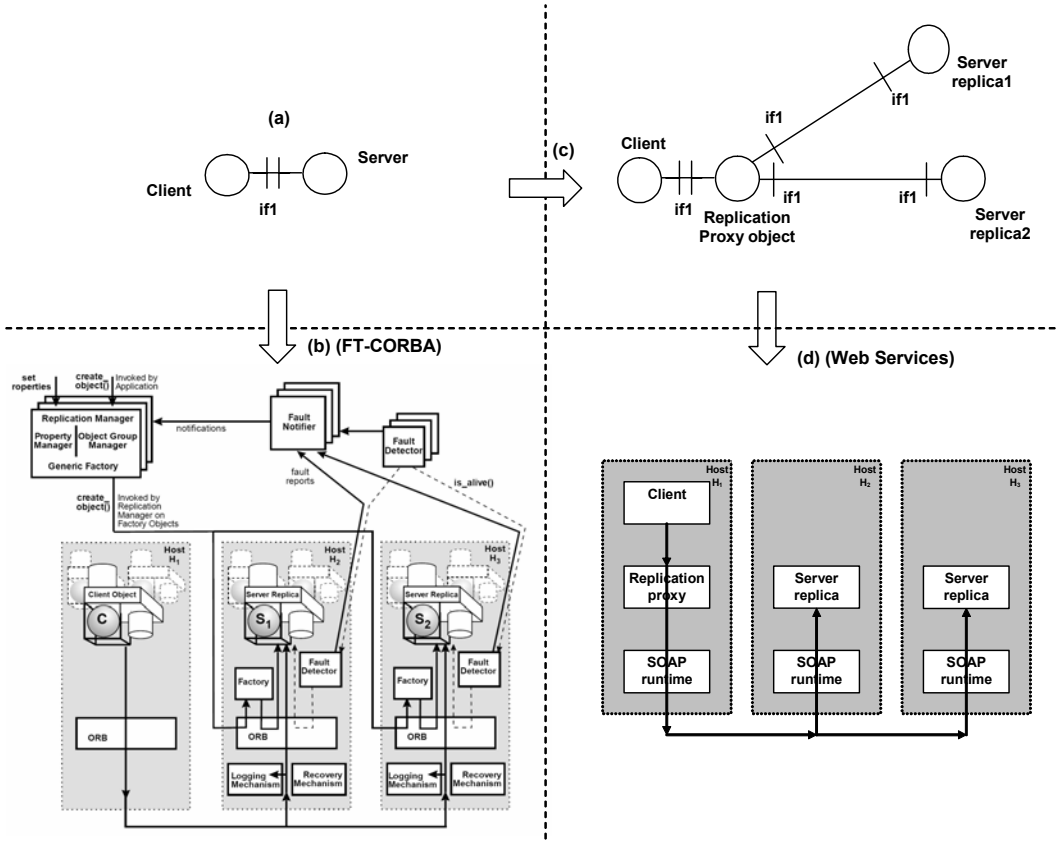



**Figure 5 Alternative mappings for abstract platform with replication transparency**

## 6. Conclusions

The separation of RM-ODP computational and engineering viewpoints is useful to distinguish between application and infrastructure concerns. This separation can be explored recursively along a model-driven design trajectory, allowing a designer to introduce infrastructure concerns progressively towards realizations on concrete infrastructures, i.e., available middleware platforms. We have demonstrated that the computational viewpoint concepts can be suitable for our design approach if we equate the RM-ODP notion of infrastructure to that of abstract platform. An abstract platform is defined in terms of the bindings supported, the transparencies supported, and the types of QoS constraints that may be applied to interface contracts. Characteristics of this abstract platform must be established by considering the different design goals.

There is no obvious distinction between platform-independent and platform-specific concerns, and no general rule to decide what is platform-independent. The needs to reuse platforms and to handle design complexity must drive a designer's decision on the boundaries. Defining an abstract platform brings attention to *balancing* between: (i) platform-independent modelling, and (ii) platform-specific realization.

The proliferation of different abstract platforms reduces the opportunities for large-scale reuse of platform-independent models and transformations. This calls for agreement on a small number of abstract platforms that are, to a great extent, application-domain-neutral and platform-independent. Ideally, a reference architecture with a small set of canonical abstract-platform-elements should be used to compose abstract platforms that suit the needs of particular projects. We intend to define such a reference architecture, based on concepts of the computational viewpoint of the RM-ODP.

Using a well-founded reference model (RM-ODP) to refer to abstract platform enables agreement on the concepts for the description of abstract platforms, and may prove to be an initial step towards a comprehensive framework for the definition of abstract platforms.

## Acknowledgements

This work is part of the Freeband A-MUSE project. Freeband (http://www.freeband.nl) is sponsored by the Dutch government under contract BSIK 03025. This work is also partly supported by the European Commission in context of the MODA-TEL IST project (http://www.modatel.org).

## References

[1] D. Akehurst, J. Derrick, A.G. Waters. Addressing Computational Viewpoint Design, in: Proc. 7th IEEE Intl. Enterprise Distributed Object Computing Conference (EDOC 2003) (IEEE Computer Society, Los Alamitos, CA, Sept. 2003).

[2] J.P.A. Almeida, M. van Sinderen, L. Ferreira Pires and D. Quartel, A systematic approach to platform-independent design based on the service concept, in: Proc. 7th IEEE Intl. Enterprise Distributed Object Computing Conference (EDOC 2003) (IEEE Computer Society, Los Alamitos, CA, Sept. 2003) 112-123.

[3] J.P.A. Almeida, R. Dijkman, M. van Sinderen, and L. Ferreira Pires, On the Notion of Abstract Platform in MDA Development, in: Proc. 8th IEEE Intl. Enterprise Distributed Object Computing Conference (EDOC 2004) (IEEE Computer Society, Los Alamitos, CA, to appear Sept. 2004).

[4] J.P.A. Almeida, M. van Sinderen, L. Ferreira Pires and M. Wegdam, Handling QoS in MDA: a discussion on availability and dynamic reconfiguration, in: Proceedings of the Workshop on Model Driven Architecture: Foundations and Application (MDAFA) 2003, CTIT Technical Report TR–CTIT–03–27, University of Twente, The Netherlands, June 26-27, 2003, 91-96.

[5] G. Arango, Domain Analysis: from Art Form to Engineering Discipline, in: ACM SIGSOFT Software Engineering Notes, Vol. 14 , No. 3, May 1989, 152-159.

[6] G. Blair and J.B. Stefani. Open Distributed Processing and Multimedia. Addison Wesley, 1997.

[7] M. Endler and D. Schmidt (Eds.). Proceedings of the ACM/IFIP/USENIX International Middleware Conference 2003, in: Lecture Notes in Computer Science. Springer-Verlag, Heidelberg, Volume 2672 / Jan. 2003.

[8] The Institute of Electrical and Electronics Engineers (IEEE) Standards Board. Recommended Practice for Architectural Description of Software-Intensive Systems (IEEE-Std-1471- 2000), Sept 2000.

[9] ITU-T / ISO, Open Distributed Processing - Reference Model – All Parts, ITU-T X.901-4 | ISO/IEC 10746-1 to 10746-4, Nov. 1995.

[10] ITU-T / ISO, Open Distributed Processing - Reference Model - Enterprise Language, ITU-T X.911 | ISO/IEC 15414:2002, Oct. 2001.

[11] ITU-T, Recommendation Z.100 – CCITT Specification and Description Language, International Telecommunications Union (ITU), 2002.

[12] ITU-T / ISO, Use of UML for ODP system specifications, ITU-T X.906 | ISO/IEC 19793 Working Draft, May 2004.

[13] Object Management Group, Model driven architecture (MDA), ormsc/01-07-01, July 2001.

[14] Object Management Group, Common Object Request Broker Architecture: Core Specification, Version 3.0, formal/02-12-06, Dec. 2002.

[15] Object Management Group, CORBA Component Model, v3.0, formal/02-06-65, July 2002.

[16] Object Management Group, MDA-Guide, V1.0.1, omg/03-06-01, June 2003.

[17] Object Management Group, UML 2.0 Superstructure, ptc/03-08-02, Aug. 2003.

[18] Object Management Group, UML Profile for Enterprise Distributed Object Computing, ptc/02-02-05, Feb. 2002.

[19] Object Management Group, Unified Modeling Language (UML) Specification: Infrastructure, Version 2.0, ptc/03-09-15, Sept. 2003.

[20] J.R. Putman, Architecting with RM-ODP, Prentice Hall, USA, 2001

[21] World Wide Web Consortium, SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation, June 2003, available at http://www.w3.org/TR/soap12-part1

[22] World Wide Web Consortium, Web Services Description Language (WSDL) 1.1, W3C Note, March 2001, available at http://www.w3.org/TR/wsdl

# Applying Model-Driven Development to Business Systems using RM-ODP and EDOC

Yoshihide Nagase
*Technologic Arts Inc.*
*yoshi@tech-arts.co.jp*

Daisuke Hashimoto
*Technologic Arts Inc.*
*hashimoto@tech-arts.co.jp*

Miwa Sato
*Technologic Arts Inc.*
*msatoh@tech-arts.co.jp*

## Abstract

*Improving development efficiency and maintainability for business systems requires a seamless development process, and both RM-ODP and MDA play a key role to this end. This paper shows our Model-Driven Development process in building business systems using RM-ODP and UML Profile for EDOC, with a case study of Electronic Health Record system models, and discusses several issues related to RM-ODP standard.*

## 1. Introduction

As business systems are getting complex and large in recent years, their development efficiency and maintainability need to be improved. Especially, efforts to Total Optimization by SCM (Supply Chain Management) have a great influence on the way business systems are developed. To conduct Total Optimization, various information systems introduced to enterprise and supply chain need to collaborate with each other. In order to develop such information systems, it is required to model business processes and identify the roles of those systems in the entire business. Moreover, certain mechanism is necessary to transform the models seamlessly into implementation, which makes systems development more efficient with traceability among models.

## 2. MDA and RM-ODP Viewpoints

MDA, advocated by OMG (Object Management Group), is the technology that seamlessly reflects models to implementation. In MDA, models are developed from three perspectives: CIM (Computation Independent Model), PIM (Platform Independent Model), and PSM (Platform Specific Model). MDA is an abstract framework, thus concrete development processes are required for realization of MDA.

This paper shows overview of our Model-Driven Development process using RM-ODP [1] framework and notations defined by UML Profile for EDOC[2] (referring to it as EDOC from now on).

Our process uses RM-ODP Viewpoints as follows: business models are developed in Enterprise Viewpoint, information models in Information Viewpoint, and component models in Computational Viewpoint. These models are CIMs and PIMs in MDA. System architectures are defined in Engineering Viewpoint, and mapping rules are defined in Technology Viewpoint. In this development process, EDOC notations with defined semantics are used to define Enterprise, Information, and Computational Viewpoint specifications, since modeling elements, such as "process," "entity" and "component" required for describing those viewpoint specifications, were already standardized in the EDOC standard.

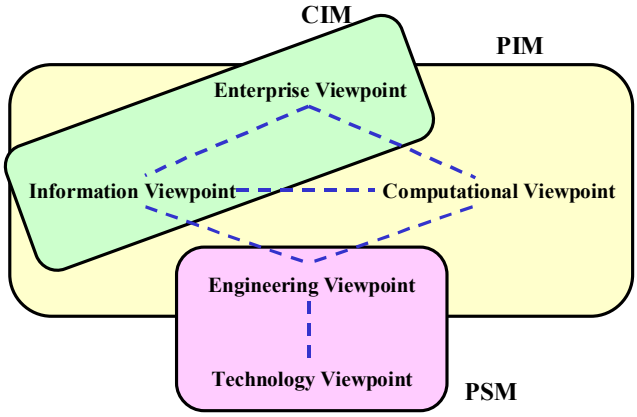



**Figure 1. Relation between MDA and Viewpoints of RM-ODP**

In the following sections, Electronic Health Record system models[3][4] are used as a case study for applying this development

[1] The standard frameworks for distributed object systems, standardized by ISO/IEC and ITU.
[2] The set of extended UML for distributed object systems, standardized by OMG It extends notations for components and business processes.
[3] The Enterprise Viewpoint part of these models are an accomplishment of "The

process. This case study is an accomplishment of a project formed by two organizations: JAHIS, whose focus is on establishing approaches for developing standard components for Electronic Health Record systems, and INTAP, whose focus is on interoperability among enterprise systems with RM-ODP. Japanese government has been involved in and funds the project.

## 3. EDOC

EDOC includes a set of following profiles and those meta-models that extend standard UML for enterprise distributed object computing environments.

### 3.1 Component Collaboration Architecture (CCA) Profile

CCA Profile is a profile used to define computational components and interactions between those components. In this profile, Process Component is defined as a fundamental functional component. In interfaces of a Process Component, three kinds of ports (Flow, Protocol, and Operation) can be defined. With those ports, three kind of interactions can be specified: data flow interaction (simple data flow in and out via Flow Port), protocol flow interaction (two-way interactions via Protocol Port), and operational flow interaction (call/return type of interaction via Operation Port).

### 3.2 Entity Profile

Entity Profile is a profile used to define entity's structure of the target domain. Entity is a specialization of Process Component. Entity Data represents an aspect of Entity's data structure. Entity Data can have primary key and foreign key like relational database.

### 3.3 Business Process Profile

Business Process Profile is a profile used to define business processes. This profile extends CCA Process Component to define Business Process, Compound Task, and Activity etc., which together provide necessary semantics and notations to represent enterprise viewpoint process models. An Activity is a work to be done to complete a process, and may be associated with one or more of three roles, which are Performer, Artifact and Responsible Party. Compound Task is a container of Activities, and Business Process is the outermost container of the composition.

### 3.4 Event Profile

Event profile is a profile used to define business processes driven by business events and the mechanism of the state transition for business entities.

### 3.5 Relationship Profile

Relationship Profile is a profile used to define clearer relationship between model elements.

## 4. Enterprise Viewpoint

In the Enterprise Viewpoint, target business models and system requirements are modeled. The most important artifacts of this Viewpoint are business processes. Organizing business processes in the entire supply chain and enterprise clarifies the role of information systems. The first thing to do in the Enterprise Viewpoint is to define a scope for the system and divide it into smaller ones for a unit called Community. It is the unit to organize the scopes with their purposes. Defining Communities can make the size of the scope appropriate. Next, procedures for accomplishing purposes of each Community are defined as business processes. With Business Process Profile and Event Profile of EDOC, event-driven and non-event-driven business processes can be described, and information and functions used in the business process as well as Responsible Parties of Activity can be defined.

Modeling language with semantics of business processes was required to define business processes. Since business process semantics was not included in standard UML, we adopted EDOC for process modeling, instead of extending UML independently by ourselves. Therefore, Business Process Profile of EDOC was used in the development of Enterprise Viewpoint specification. Should we start this activity today, we will need to carefully watch and choose a right standard from BPDM, UML for ODP, and UML2.0 etc.

In this viewpoint, we did not make much use of developed policy statements. The issue was lack of policy statement language with clear syntax, semantics, and grammar. Expected are the emergence of policy statement language, and the structuring rule for policies with other enterprise model elements in the enterprise viewpoint specification.

---

Development of Electronic Health Record System through Standardizing Components" as a special science research theme awarded by the Ministry of Health, Labor and Welfare in the fiscal year 2002.

4 The scale of those models includes 12 communities and 32 business processes.

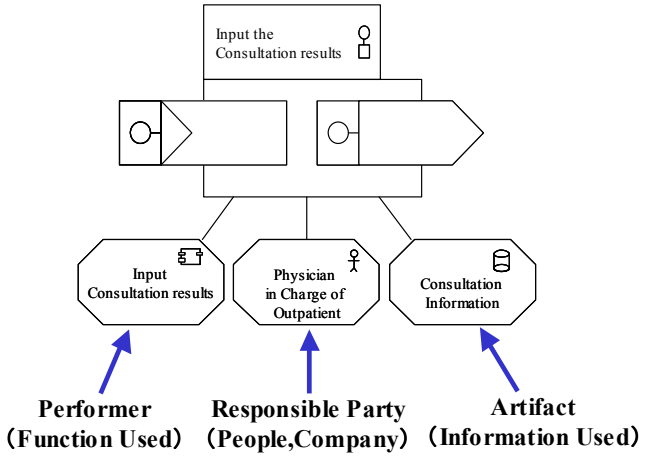



**Figure 2. Example of Activity**

## 5. Information Viewpoint

In the Information Viewpoint, information referred to and processed by the systems is modeled. Our development process defines that the details of information used in the business processes should be analyzed. The resulting information model elements of the system are described using Entity Profile in the EDOC. Later in this paper, the use of CCA Profile to describe Computational Viewpoint specification is explained. In order to integrate functional components derived from Enterprise Viewpoint specification with Entity Components from Information Viewpoint specification to complete the Computational Viewpoint specification, it is necessary to componentize Information Model with EDOC.

The use of the Entity profile enables description of primary key and foreign key of entity implemented using relational database, and define access path to the target information by creating Entity Components. Constructing information as components is effective in preventing reduction of performance, since it identifies the gate of information to control the number of remote accesses.

In this viewpoint, we did not consider dynamic addition or deletion of model elements. However, if dynamic change is included in Enterprise Viewpoint Model, such as dynamic creation of a Community with new roles introduced, information viewpoint model should also accommodate the corresponding dynamic changes within the model.

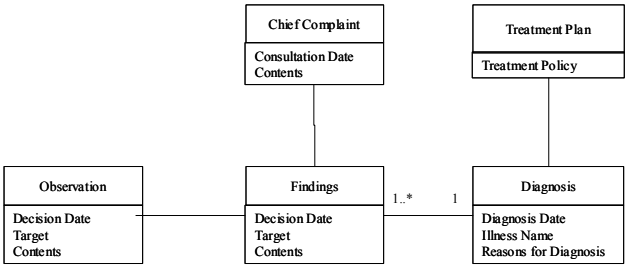



**Figure 3. Example of Information Model (Consultation)**

## 6. Computational Viewpoint

In the Computational Viewpoint, functions of systems are modeled. Our process defines that the details of functions used in the business processes (i.e. performers and artifacts) and entities should be analyzed, and the functions of systems are described as Process Components of CCA Profile in the EDOC. Therefore Computational Objects in RM-ODP are represented as Process Components of CCA.

If collaboration exists among enterprises and/or among information systems, it is described as interaction among Process Components. Note that interfaces of components are discovered based on analysis of Information Viewpoint.

In the EDOC, collaborations are described by connecting components through Ports. If the functions of systems require persistent information (database information), the Process Components are connected to the Entity Components. Operation procedure of the connected components (both process and entity) is defined as Protocol and described using State Machine diagrams of UML.

It should be noted that in RM-ODP the basic unit for encapsulation is object, and the interaction between objects are made by Signal, Operation, and Flow. Although we believe CCA Process Component can represent Computational Object in pragmatic way, it would be preferable to have necessary modeling concepts for components and interactions between them (messaging) in RM-ODP itself (a discussion on the use of object versus component is described in 9.1).

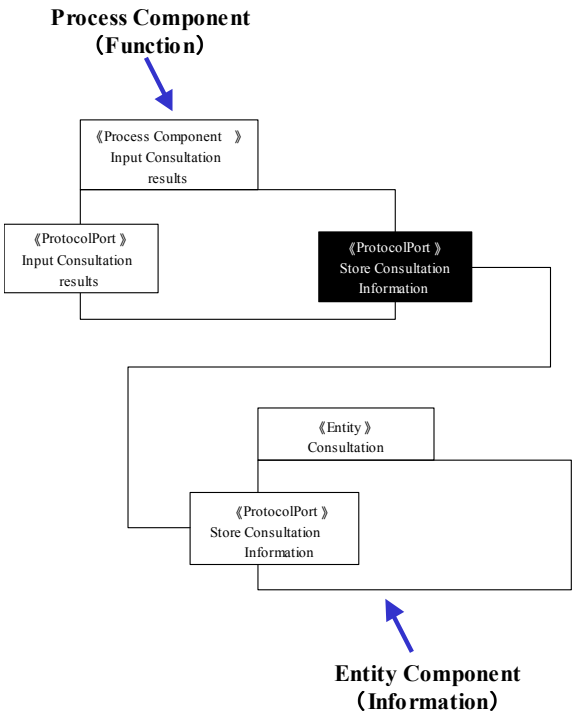



**Figure 4. Example of Component Model**

## 7. Engineering and Technology Viewpoints

In the Engineering Viewpoint, system architectures are designed. The design includes deployment configuration of components, transaction, and security.

It should be noted that there are choices in mapping Component Model onto platforms. One choice is on platform styles, and another choice is on configuration of nodes. Figure 5 was our choice for pilot implementation, but there were other possibilities, e.g. client node could host only GUI application and network interface components, and all other components may be hosted on one or more server nodes. Also the same component model could have been mapped onto web services platform or CORBA platform.

Another observation is that it is this viewpoint that shows portions of human-system interaction with Client node box. The model of human-system interaction should also be described in relevant viewpoints in addition to the Engineering viewpoint model.

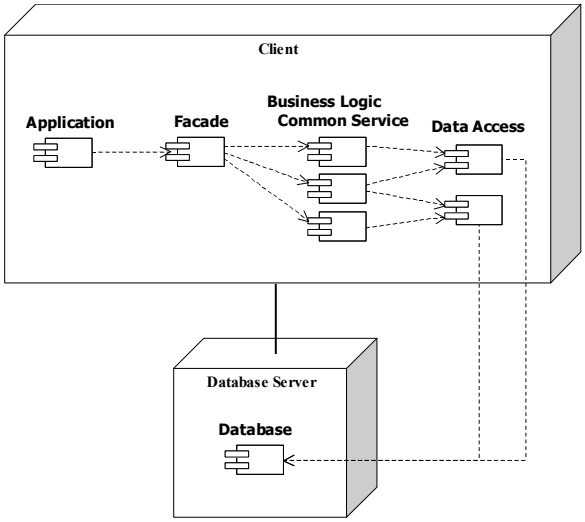



**Figure 5. Example of Deployment Configuration of Components**

In the Technology Viewpoint, PIM as the artifact of all the processes is mapped to a specific platform to derive PSM from it. The following example is one of mapping rules of EDOC to .NET. The mapping rules were used in the pilot development of Electronic Health Record system.

| EDOC (PIM) | .NET (PSM) |
|---|---|
| Process Component 《ProcessComponent》 | Mapped to .NET assembly. Select assembly type according to requirement such as deployment, performance, and security. |
| Protocol 《Protocol》 | Mapped to interface. |
| Interface 《Interface》 | Mapped to interface. |
| Protocol Port 《ProtocolPort》 | Mapped to interface. |
| Operation Port 《OperationPort》 | Initiator: mapped to method call. Responder: mapped as interface method. |
| Flow Port 《FlowPort》 | In case of flow port included in operation port: Initiator: mapped to argument of method. Responder: mapped to returned value of method. |
| Entity Component 《Entity》 | Mapped to .NET component using ADO.NET. |
| Entity Data 《EntityData》 | Mapped to ADO.NET DataSet defined by XML schema. |
| Key 《Key》 | Mapped to constraint defined by XML schema. |
| Foreign Key 《ForeignKey》 | Mapped to constraint defined by XML schema. |

**Table 1. Example of Mapping Rules (EDOC to .NET)**

In Table 1, interfaces provided by the Process Components and various Ports included in the components are mapped to interfaces of .NET components and their methods. The Entity Components are mapped to .NET components that use functions of ADO .NET. Remote accesses among components may be required depending on deployment configuration of components in Engineering Viewpoint. In this case, Web services and remoting functions of ASP .NET should be used to realize the remote accesses.

## 8. Viewpoint correspondence

To realize the seamless development and to ensure the traceability, we need to elaborate on viewpoint correspondence. In RM-ODP and Enterprise Language standard, viewpoint correspondences are defined but are not very useful in practice. In EDOC, because of its design, Business Process Profile, Entity Profile, and Event Profile are defined as specializations of CCA Profile, and thus there exists stronger correspondence between those and CCA Profile. Once the mapping from those to CCA Profile is done, and the final model is represented in CCA Profile, the resulting model will be considered as a Computational Viewpoint Model. In our development process, EDOC/CCA-based viewpoint correspondences are applied to those viewpoint specifications.

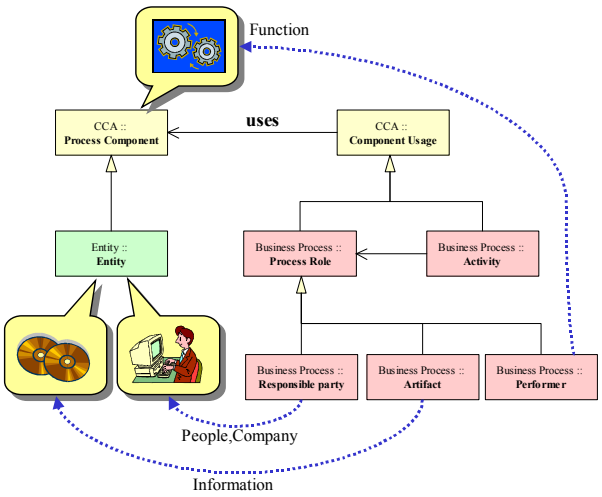



**Figure 6. Relationship of EDOC Profiles**

## 9. Discussion

Based on the experience of the above explained project, it is the our belief that the following issues should be considered at the workshop.

### 9.1 Computational Object vs. Component

In this development process, Computational Object is represented as a Process Component. However, object and component are not completely the same. For instance, according to reference [11], component's characteristics are described as "A coherent package of software implementation that 1) has explicit and well-specified interfaces for the services it provides; 2) has explicit and well-specified interfaces for services it expects from others; and 3) can be composed with other components, perhaps customizing some of their properties, without modifying the components themselves. As a consequence of these properties, a component can be independently developed, delivered, and deployed as a unit."

Computational Object of RM-ODP satisfies the characteristics 1) and 3), but not the characteristics 2). In addition, some components may work only within specific platform (or framework), and that platform (framework) might require certain components as prerequisite, or might require specific interface type for components. Computational Viewpoint is a viewpoint where computational aspects of the systems are the major focus, and we believe that an ODP system comprising loosely coupled objects, which require messaging based interactions, plays an important role in today's internet world. It is, therefore, desirable to have not only object-based computation, but component-based computation. In this case, we propose to introduce a basic concept of component into Computational Language ([3] instead of [2]). Note that this proposal has close relationship with the next proposal regarding messaging.

### 9.2 Communication by message

Communication or interaction between objects in Computational Viewpoint is basically synchronous with RPC style. However, in the real world (e.g., in business-to-business transactions), asynchronous communication, or message-based interaction, is in fact used widely. If we only need to provide modeling capability for asynchronous communication, the discussion of property for the Binding Object will work. The difference between synchronous communication and asynchronous communication in Computational Viewpoint appears when describing a behavior of a client. While a client is blocked by issuing requests in synchronous communication, a client is allowed to continue processing in asynchronous communication even after issuing requests. The client in this case may need to monitor the status of the result in parallel to its processing in a different thread. For instance, in the case of Web Services, service provider may make two types of Web Services descriptions public: synchronous and asynchronous. And, the behavior of the client is quite different. The following is the issues for RM-ODP.

1) Computational Language provides interfaces for objects which provide services, but does not provide interfaces for object requiring services. The behavior of the client is dependent on this "interface for object requiring services." and it is hard to specify the client's behavior without having this concept.

2) There should be a standard way of describing structure of activities at the user side of interfaces. If activity structure section of [2] is assumed for this purpose, appropriate statement

should be added in Computational Language.
3) At this interface description level, there may be a necessity to introduce message management interface such as handling messages in a queue.

### 9.3 Concept for composite interaction

Three kinds of interactions are defined in Computational Language: Signal, Operational, and Flow. However, the concept of composite interaction, combining these different types of interactions, is not defined clearly. This concept will be useful if the interaction by two parties, like two companies in a typical business-to-business business process scenario, is represented as interactions between two Computational Objects, since the interaction may contain all the interaction patterns. It might be possible to consider Flow as the concept for this purpose, if you interpret the definition of Flow in [3] literally. If this is the case, the standard should be clear enough to state this. If this is not the case, the concept for composite interactions should be added in the Computational Language. Note that EDOC/CCA's model element called Protocol manages the composite interactions, and UML2 has Protocol State Machine as a similar model element.

### 9.4 Dynamic evolutions of models

The dynamic evolutions of models means that model elements may be added or changed or deleted at some point to evolve into the model in the next phase. It may be viewed as a part of the lifecycle of models or versioning of models. For example, in the Enterprise Viewpoint Language, dynamic creation and deletion of communities are described. Other possibility includes dynamic addition of Enterprise Objects, Roles, Processes, and so on. If Enterprise Viewpoint Model evolves in such a way, corresponding viewpoint models should also evolve. In Information Viewpoint, Information Object might be added or deleted, and defined schemata may also evolve. In Computational Viewpoint, the whole model might change. It will certainly impact the engineering and technology viewpoint models. We will need to discuss:
1) how these evolutions or changes (transition) can be described in each viewpoint, and
2) what concepts in which viewpoint is required to achieve this.
If we were successful, we may apply the result to "as-is model" and "to-be model" in Enterprise Architecture today.

### 9.5 Human-System interactions

In the Viewpoint Languages ([3] and [4]) today, there seems to be no place for describing Human-System Interaction. However, [2] defines "Perceptual Reference Point" under "Class of Reference Points" (see [2], section15.3.2). We believe it important to clarify the position of the standard. The discussion we would like to have is whether we would like to introduce new concepts around human-system interactions. Even if the result of it leads to take no action, there is a need to provide a guideline or an official statement on this point. If we could have concept(s) for this purpose in viewpoint languages, the following may be a possible candidate where the concept(s) may contribute to.
1) Specification of Human-System interactions for the system providing similar services through multi-channels (Web, telephone, fax, e-mail, and letter)
2) Screen design and screen flow for a Web-based application
Note that there is a similar issue regarding "Interchange Reference Point."

### 9.6 Policy language

The current policy related concepts are those of meta-model level concepts, and may be used as policy categories, but those are not detailed or concrete enough to apply in the real world specifications. The work done in network management area may be of interest to us. The followings are several modeling elements (minimum) to be considered.

The policy will be initially described in natural languages. Therefore, the following elements may be required, as a minimum, to create a complete policy statement with existing policy concepts: subject, verb, object, and condition. If we tried to associate with Enterprise Viewpoint Language concepts, the following may apply to some cases.
1) subject <-Enterprise Object | Role | Community
2) verb <- Action including Obligation and so on| Process
3) object <- Enterprise Object | Role | Community
4) condition <- Predicate | Guard Condition
Policy statement example:
"If an emergency patient is brought into the hospital (Condition), a doctor on duty (subject: Role Doctor fulfilled by Enterprise Object Person) has an obligation (Obligation) to make a diagnosis of the patient (verb: Action 1, object: Role Patient fulfilled by Enterprise Object Person) or find a suitable doctor to request a diagnosis for the patient (verb: Action 2, object: Role Doctor fulfilled by Enterprise Object Person)."

## 10. Summary

### 10.1 Model-Driven Development

Our concrete process worked well to develop business systems by Model-Driven Development, resulting in successfully applying our process to an Electronic Health Record system. It turned out that applying development frameworks of RM-ODP enabled seamless development from business models to program codes. Especially in developing business systems that aim for Total Optimization, this framework is certainly the best since the scope of collaboration can be considered as distributed objects. The use of EDOC also enables to describe various aspects of business systems in detail. As EDOC has great affinity for RM-ODP, it is the most appropriate modeling language in building CIMs and PIMs. Since this case study of Electronic Health Record system is implemented as the pilot system, our next goals are to use our process to implement an Electronic Health Record system of practical scale, and to apply our process to systems with other domains for its refinement.

### 10.2 Possible revisions to the standard

We have identified several issues for RM-ODP standard, and included some of our suggestions in this paper. It is our hope that those issues be considered, discussed, and resolved at the workshop, or passed to ISO/IEC and ITU-T RM-ODP group as a part of issues list for RM-ODP revision work.

## Reference


[1] ISO/IEC IS 10746-1, *Open Distributed Processing -Reference Model: Overview* 1998.

[2] ISO/IEC IS 10746-2, *Open Distributed Processing -Reference Model: Foundations*, 1996.

[3] ISO/IEC IS 10746-3, *Open Distributed Processing -Reference Model: Architecture* 1996.

[4] ISO/IEC 15414, *Open distributed processing - Reference model -Enterprise language*, 2002

[5] UML Profile for EDOC Part I, Document number: ptc/2004-02-01

[6] UML Profile for EDOC Part II, Document Number: ad/01-08-20

[7] Model Driven Architecture, Document number: ormsc/01-07-01

[8] MDA Guide, Document number: omg/03-06-01

[9] JAHIS Web Site, http://www.jahis.jp/english/english.html

[10] INTAP Web Site, http://www.net.intap.or.jp/e/

[11] D.F. D'Souza and A.C. Wills, *Objects, Components, and Frameworks with UML (The Catalysis Approach)*, Addison-Wesley, 1998

[12] Moore, B., Ellesson, E., Strassner, J. and A. Westerinen, *"Policy Core Information Model - Version 1 Specification"*, RFC 3060, February 2001.

[13] Moore, B., Ed., *"Policy Core Information Model Extensions"*, RFC 3460, January 2003.

[14] Distributed Management Task Force, Inc., *"DMTF Technologies: CIM Standards CIM Schema: Version 2.8"*

[15] Distributed Management Task Force, Inc., *"Common Information Model (CIM) Specification: Version 2.2"*, June 14, 1999

# Architecting Frameworks for Specific Applications with RM-ODP

Ana Paula Gonçalves, Sandro Antônio Vicente, Dib Karam Jr, Moacyr Martucci Jr
*Computing Engineering and Digital Systems Dept – Escola Politécnica da USP*
*apaulacg@uol.com.br, sandro.vicente@poli.usp.br, dib@poli.usp.br, moacyr.martucci@poli.usp.br*

## Abstract

*Today, distributed systems are commonly used in business enterprises in practically all market sectors. But such systems are characterized by their huge complexity due to physical distribution, lack of synchronization, heterogeneity, external parties and the very business logic related to the system itself. RM-ODP appeared as an interesting resource to assist the designing of architectures for distributed systems, providing means to capture business needs, distributed processing systems architecture, semantics of processing, and choice of technologies. This paper presents three different experiences in modeling architectures using RM-ODP: a proposal for use of RM-ODP for the development of convergent applications, a generic architecture for CRM systems and a framework based on ODP for the integration among different computer architectures.*

## 1 Introduction

Nowadays, distributed systems became mature enough to be commonly used in business enterprises in practically all market sectors. Besides their business complexity, such systems are characterized by physical distribution, lack of synchronization and heterogeneity due to the fact that they are composed of a plethora of different applications and devices. In addition, such systems are also affected by external elements, such as telecommunication networks, 3rd party systems, etc.

Hence, when a distributed system is being designed, it is necessary to assure that its architecture will provide the right levels of service, supporting interoperability, scalability, security, heterogeneity, and all other aspects that characterize such systems.

RM-ODP (Reference Model – Open Distributed Processing) appeared as an interesting resource to assist the designing of architectures for distributed systems, providing means to capture business needs, distributed processing systems architecture, semantics of processing, and choice of technologies, all in a consistent and complete manner. It focuses on how to capture the components, their interrelationships, and formal semantics of processing aspects of an open distributed processing system.

The paper objective is trying to show how the RM-ODP can be considered a complete modeling tool, presenting an overview for three generic cases. Each case focuses a different set ofviewpoints, chose according its importance in application. The complete set of the viewpoints were used considering the three presented cases together.

These experiences in modeling architectures addressing specific applications domains carried out by integrants of the Computing Engineering and Digital Systems Dept of Escola Politécnica da USP (University of Sao Paulo), which have been driving early researches in this area in Brazil.

This paper is organized as follows. Section 2 presents a proposal for use of RM-ODP aiming the development of convergent applications. Section 3 presents a generic architecture for CRM (Customer Relationship Management) systems, which fits CRM strategies into the ODP (Open Distributed Processing) enterprise language. Section 4 outlines a framework based on ODP for the integration among different computer architectures. Finally, section 5 concludes the paper.

## 2 Designing technologically convergent applications using RM-ODP

This section presents a proposal for use of RM-ODP aiming the development of convergent applications. But it is important first to clarify the meaning of the term “convergent”.

The term technological convergence is often mistaken for interoperability. The difference among them is that “interoperability” means capability to work with another autonomous systems or applications. Technological convergence means adaptation of services to different communications medias through the use of networks and terminals designed to bear such services, transparently providing users with access to these services’ information and applications. So these services are carried out using

any network, any communications channel and providing a coherent human-interface with appropriate quality. This demands fundamentally capabilities of mobility, portability of applications and content, and interconnection and interoperability among platforms and operators. Any application involving several technologies, such as: digital television, mobile Internet, videoconference, telephony, interactive broadcasting, etc, can be considered a convergent application [1].

An appropriate solution in order to develop convergent applications employs the model of distributed computing using the Internet, client-server architecture in a decentralized networked environment, and independent and autonomous devices. Generally, this distributed computing model uses mainly the TCP/IP (Transmission Control Protocol - Internet Protocol protocol, for the communication among devices, using different architectures, operating systems and applications. Distributed object architectures provides suitable means to apply the model of distributed computing, enabling the transparent integration of distributed services upon software architectures, hardware platforms and networks [2][3].

Figure 1 presents an overview of architecture of convergent applications based on three different technologies: the Internet, mobile telephony and digital television. The integration among these three technologies is possible only through the use of standardized protocols and distributed objects. Thus, next section presents how convergent applications can be developed using the RM-ODP.

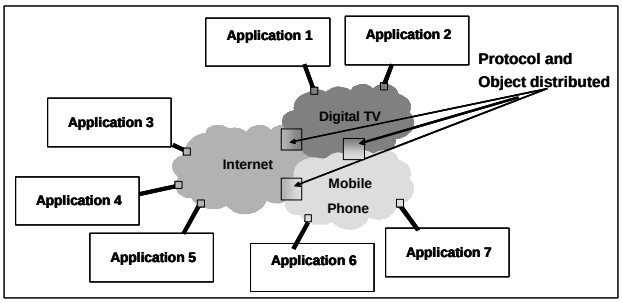



Figure 1. General view of architecture of convergent applications based on three different technologies

The following subsections present an overview of architecture for convergent applications using the computational, engineering and technology ODP viewpoints. The enterprise and information viewpoints are not addressed in this article because it aims to propose a generic architecture, able to be applied over different enterprise and information models.

## 2.1. Computational viewpoint

The computational viewpoint is responsible for the functional decomposition in terms of objects and their computational interfaces, which must be specified for the development of convergent applications [4]. Figure 2 presents the objects and interfaces and they are detailed as follows.

- User object: represents communication channel through which a user may use to send or receive information and services. Examples of instances of this object are: PDAs (Personal Digital Assistants), mobile telephones and digital television. This object has interface with Connection Object;
- Connection object: is the object responsible for connection services among the user object and the service required, according to the type of communication channel. Examples of instances of this object are: Internet connection via TCP/IP, mobile phone connection via WAP (Wireless Application Protocol), digital television connection using image, sound and data compression and decompression techniques through the use of DVB-T (Digital Video Broadcasting - Terrestrial) standard. This object has interface with Convergence Services Objects;
- Convergent services object: these objects represent convergent services, so they can be considered an application framework integrating different convergent services and obtaining services and information from different locations.

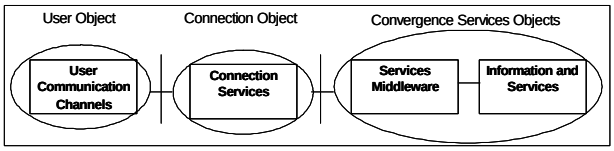



Figure 2. Computational view of objects for convergent applications

## 2.2. Engineering viewpoint

The engineering viewpoint model provides support for the execution of the computational model [4]. Figure 3 presents a simplified architecture in the engineering viewpoint, detailed as follows.

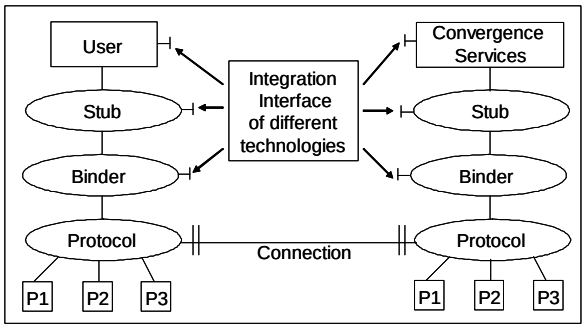



Figure 3. Simplified architecture of convergent applications in the engineering viewpoint

When an *user* object requires a service or information, a *stub* object will provide functions to support transparency in the request of a service and its response. The *binder* object verifies compatibility among interfaces and keeps the connection integrity of the service request and its response. The protocol object interacts with protocols of the other different convergent services. For example, in figure 3, protocols like P1, P2 and P3, transparently obtain the information or service from the binder and stub objects of the instances of the *convergent services* objects. The integration among Internet services, mobile telephony and digital television, for example, is possible through the use of interfaces concerning different technologies. An example for the use of this architecture would be a request of a service from a digital TV device (*user* in figure 3) that would enable a content search in a database server (*convergence service* in figure 3) using the Internet (*connection* in figure 3), whose answer would be delivered to the user's mobile phone (*user* in figure 3).

### 2.3. Technology viewpoint

The technology viewpoint specifies the software and hardware components of the application [4]. Such technologies for convergent applications were proposed in light of studies on well-grounded technologies, availability of tools to support development and ease of development.

The platform generally adopted is Java, because it provides APIs (Application Programming Interfaces) and facilities for the development of applications using different communications medias and devices, such as the following:

- Internet integration interface: J2EE (Java 2 Plataform Enterprise Edition) platform for the development of distributed objects allows both the use of an IDL (Interface Definition Language) with CORBA (Common Object Request Broker Architecture) and XML (eXtensible Markup Language) with SOAP (Simple Object Access Protocol);
- Mobile telephone integration interface: use of the technology MIDP (Mobile Information Device Profile). The main characteristics of MIDP concerning software are the use of specific Java APIs for limited devices, as for example: *java.lang.** classes, *java.util.** classes, *java.io.** classes, HTTP (hypertext transfer protocol) 1.1 protocols and HTTPS (hypertext transfer protocol security) X.509 protocols. Characteristics concerning hardware are the screen size of 96x64, 1-bit intensity, black and white, 4,096 colors or touch-screen enabled;
- Digital television integration interface: assuming the use of the European standard DVB-T, the MHP (Multimedia Home Platform) can be employed. Its main characteristics concerning software are the use of Java APIs like, for example: Personal Java, java TV API, Java Media Framework and DVB API extensions. Characteristics concerning hardware are MPEG-2 (Moving Picture Experts Group – 2) reception, screen resolution of 720x576 pixels and "true color" model.

Figure 4 presents the technology layers proposed for the development of convergent applications. The layer that represents the facilities of the applications' implementation is the main concern and is shown in the figure 4 with shaded blocks.

## 3 Designing a CRM architecture with ODP

In this section, an overview of a generic architecture for CRM systems is presented in light of the enterprise and information viewpoints. At first, the main idea of CRM concept is stated, as well as the concept of CRM Ecosystem [5], which is essential to drive the modeling of the CRM architecture in an enterprise context.

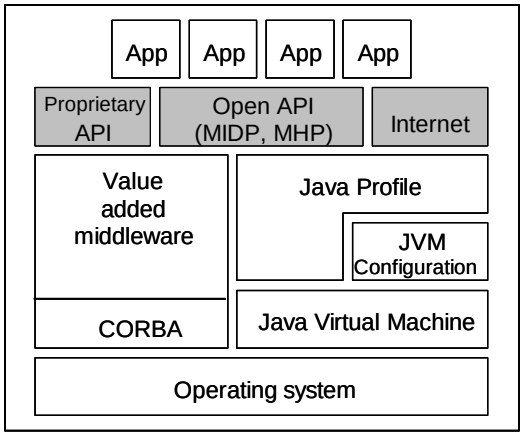



Figure 4. Technology layers for the development of convergent applications

### 3.1. Customer Relationship Management - CRM

CRM is a marketing concept whose goals are the acquirement of new customers and the loyalty of existing ones. These goals are reached establishing a friendship relationship with customers, employing one-to-one interaction, to achieve complete knowing of them, predicting their behavior and habits [6]. Technology supports one-to-one customer interaction by means of automated and semi-automated contact points providing accessibility and distribution of information to the customers [7]. Interactions among contact points and front-office applications (such as sales, marketing and customer services) implement CRM process.

CRM is implemented through the automation of horizontally integrated business processes involving front-office customer touch points via multiple, interconnected delivery channels [5]. We can distinguish three large functional groups necessary for a CRM architecture: *operational*, *collaborative* and *analytical*, where the operational CRM can be divided into front-office and back-office. Front-office functions are performed by customer service, marketing automation and sales automation applications. Back-office are performed by Enterprise Resource Planning (ERP) systems, Supply Chain Management (SCM) systems and legacy systems. Analytical CRM comprises decision systems and tools for business performance analysis such as: data warehouses, data marts and data mining tools. Collaborative CRM comprises elements used as customer channels, such as: IVR (Interactive Voice Response) devices, CTI (Computer Telephony Integration) systems, ACDs (Automatic Call Distributor), web sites, agents terminals, etc [6]. These three functional groups working together establish a CRM Ecosystem, which is depicted in figure 5.

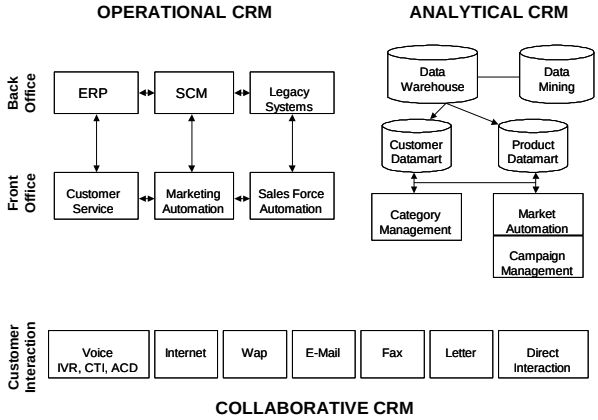



Figure 5. The CRM Ecosystem

### 3.2. Enterprise viewpoint

In this subsection, a generic architecture for CRM systems is modeled using the ODP's enterprise viewpoint [4]. This viewpoint is the basis to understand the overall structure of the CRM architecture in terms of which elements take part in the architecture and which roles they perform.

In the enterprise viewpoint, the entire CRM system is modeled by means of business objects, or enterprise objects, each one performing the roles necessary for the activities concerning those functional groups that perform the CRM process: *Collaborative* (CO), *FrontOffice* (FO), *BackOffice* (BO) and *Analytical* (AN). In addition, the customer who interacts with the CRM system is also considered a role in the enterprise viewpoint because it represents an entity external to the CRM system. Business roles and the interactions among them are depicted in figure 6 and detailed as follows.

- *Customer*: performed by objects that represent the CRM system' customers, concerning policies related to customers location, current situation, preferences, etc. Customers may interact with objects performing CO role;
- CO: performed by objects representing the applications and devices, or groups of applications and devices that interact directly with the customers, such as IVR devices, human agent workstations, web connections, etc;
- FO: performed by objects which perform activities such as: customer care services, contact

management, telemarketing and sales force automation;

- BO: performed by objects responsible for the core activities of the business where the CRM system is applied to, concerning ERP systems, legacy systems, SCM systems and operational databases;
- AN: performed by objects responsible for the analytical CRM which perform activities that give intelligence to the CRM process, providing the other functional groups with policies so that the customer expectations could be better fulfilled. This role extends to data warehouses, data mining applications and OLAP (On-Line Analytical Processing) tools.

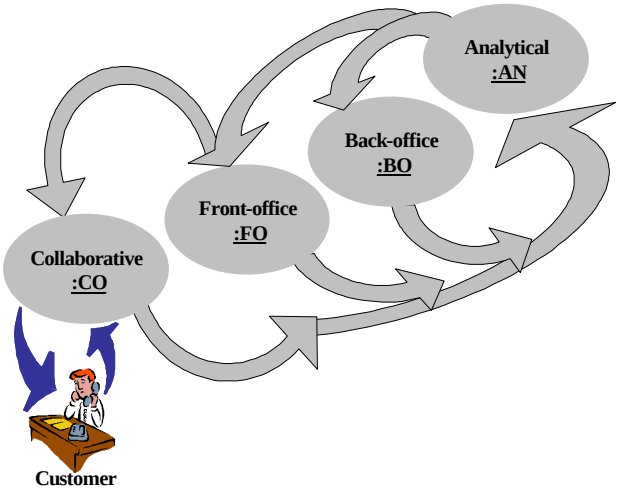



Figure 6. CRM communities in the enterprise viewpoint

The roles detailed above also define communities, each one containing sets of objects performing the same role. The interoperation among these communities is crucial to the one-to-one process and, consequently, to the CRM objectives.

So, all those communities must comprise a federation whose primary objective is to provide a one-to-one service to the customer. A service can be further modeled as a set of interactions among the CRM communities. For example, the analytical community (AN) may detect a business opportunity and request the telemarketing automation system (FO community) to contact some customers to offer a product employing a customer channel, such as: voice, e-mail or postal delivery (CO community). For the customers that accept the offer, the back-office community (BO) will process their orders. Anyway, the analytical system will process the customer responses in order to learn a little bit more about them for future contacts. So the CRM goals will be achieved.

### 3.3. Information viewpoint

In this work, the information viewpoint is used to model the general structure of information concerning the CRM system. This viewpoint may deal with the invariant, static and dynamic information schemas, but in the case of the generic architecture for CRM systems, the invariant schema is more relevant to provide a class diagram representing the structure of the information essential for a CRM process. Thus, figure 7 depicts such class diagram, which identifies the following information classes concerned with the CRM process:

- InfComponent: models information concerning the devices used in the communities CO and FO, where specifics characteristics of each of these communities are carried out by the specializations InfChannel and InfApp, detailed below.
- InfChannel: models information concerning the touch-points (or channels) with customers, such as: IVR (Interactive Voice Response) devices, telephonic branches, web-server connections, etc;
- InfApp: models information concerning front-office applications, such as help desk, sales force automation systems and market automation systems;
- InfExAgent: models external entities (probably a customer) in touch with the CRM system in a specific moment, such as during a phone call for the company's call center;
- InfInteraction: models associations involving external entities, customer channels and front-office applications, representing a well defined interaction of a customer with the CRM system;
- InfContact: models groups of inter-related interactions;
- InfCustomer: represents each customer, comprising every piece of information related to him or her;
- InfProduct: represents products and/or services offered to the customers;
- InfDeal: models the relationship among a customer, products (or services) and the contact (group of interactions) that drawn the customer to the product;
- InfCampaign: models campaigns about products, relating products to groups of customers likely to appreciate them, and involving appropriate front-office applications to offer these products to the customer.

In the context of a CRM system, the static and dynamic schemas are not so general as the invariant schema. The static schema defines specific states for each instance of the information classes identified in the invariant schema. For example, an instance of the class InfCanal, representing a telephone line, may have an attribute *status* that can be *idle*, *busy*, *disabled* and *fault*.

These values comprise states represented by the static schema.

Dynamic schemas can be used to define state transitions among the states identified in the static schema. For example, the behavior of a telephonic line can be described by a state diagram involving the states *idle*, *busy*, *disabled* and *fault*, as well as the events and conditions that trigger transitions from one state to another.

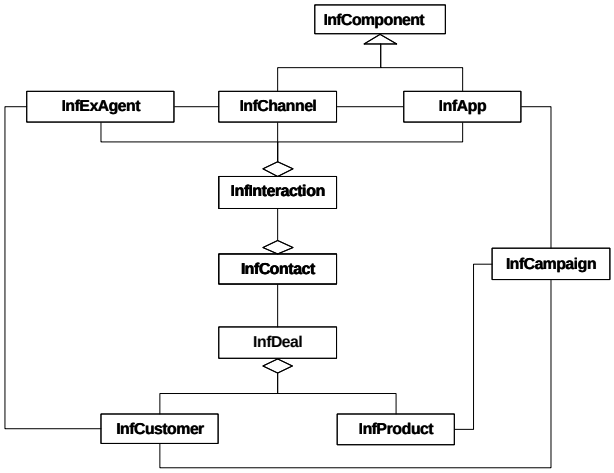



Figure 7. CRM static information schema

# 4 Middleware for integration among computer architectures

Enterprises applications have a large range of independent systems sometimes without interactions or fragile relationship [8]. On the other hand, dynamic advances in IT (Information Technology) increased complexity and customers demand for distributed information shown necessary a new way to manage IT systems.

## 4.1. Distributed processing problem

The relationship between different applications may be transparent, receiving and sending requests without new codes or additional application.

All companies have distributed processing applications without interactions allowing double effort. Today, middlewares (management, availability and basic communication layer) have functions to interact with applications available for these stand alone systems.

The companies' wish is to allow systems and databases integration within enterprises and across enterprises, collaborations, mergers, acquisitions and the Internet (a totally unstructured data source), solving problems concerning heterogeneity and distribution. It is not a wish, is a necessity around the computational world because the information is an asset more valuable than the company's facilities. By this way, accomplishing this whish is a complicate task and it is a challenge for developers and researchers [9].

In this context we are developing a multipurpose middleware, whose task is to allow integration and provide interoperability to distributed computer architectures. This middleware has been desiged to be used like an applications integrator providing interoperability between these applications. It is a necessary step for convergence between technologies.

RM-ODP brings a general architectural view for this middleware as well as global conceptual definition, analytical structure and standard specification. This section focuses on business information (information flows and structures, restrictions and standards) and computational viewpoint (it is a real need for the system).

## 4.2. Enterprise viewpoint

It outlines a middleware architecture that will integrate independent and different applications, being capable to access distributed data and execute distributed tasks. This middleware will provide information wherever, however and whenever the user request. Therefore, this middleware architecture will be a distributed programming model and will allow communication for all applications in several scenarios with flexible customization and configuration.

At the enterprise viewpoint, this architecture will install a component in existing applications that will intercept the requests and send them to the new middleware. When receiving the requests, this middleware will provide the best way available to attend it. This execution will be transparent to the users, *i.e.*, the requester and executor are not publicly visible. A given application will take part of a new system comprising several architectures. Therefore, this application will be an object for this new system with its functions and it will access and accept others applications' requests.

This architecture improves the use of a corporative system, assuring information integrity, quality of service (QoS) levels and availability.

## 4.3. Information viewpoint

By information view, the component installed in an application responsible for the interception of requests and redirection to the middleware is the request object.

Another component, also installed in the existing application, will receive the answer and will send it to the final user. It is a reception object. This mechanism is shown at figure 8.

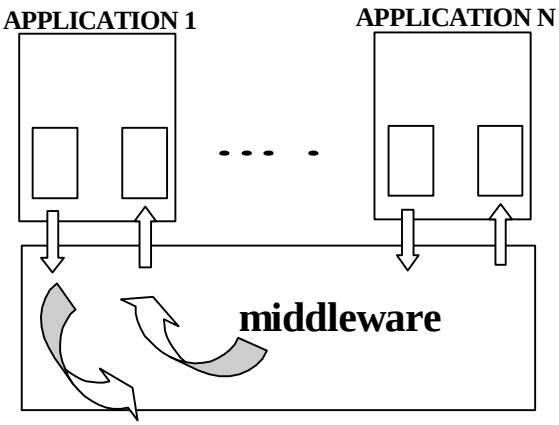



Figure 8. Information flow in the new architecture

The middleware receives, dispatches and sends the requests and answers to its users transparently. It is a way to accomplish heterogeneous management.

An existing architecture connected to this middleware will receive and send requests, integrating independent systems with different architectures without different gateways or bridges between each system.

## 5 Conclusion

The different approaches presented in this article show how RM-ODP can be properly used for the specification and modeling of distributed systems targeting different problems, which restates how RM-ODP may be malleable for architecting of distributed systems.

In the three cases presented, the studies show a weak points in RM-ODP when it was used for modeling systems where one or more blocks are legacy systems. Furthermore, for the implemetation the concern is the lack of compatibility between RM-ODP and distributed objects architectures like CORBA and J2EE.

Despite de fact that RM-ODP is not proper to formally specify an entire system, which would require to dig into its implementation details – in fact, such deed would be at least impracticable –, RM-ODP is appropriate to determine the architectural patterns necessary to drive the further development of the system, once RM-ODP modeling requires a good understanding of the interactions of the system with its environment, the elements that comprise the system, their interrelationships and the activities that must be performed, which compels the architects and designers to appropriately reason what to specify. Once a good architectural specification is ready, it is possible to provide an implementation for it using well-grounded technologies, basing on the ODP technology viewpoint.

## 6 References

[1] Presidencia Española de la Unión Europea, *El Potencial de la Convergencia Tecnológica en el Desarrollo de la Sociedad de la Información*, Colegio Oficial ingenieiros de telecomunicación., 2002.

[2] H. Balen, "Distributed Object Architectures with CORBA", *Sigs Books*, Cambridge University Press, 2000.

[3] G. Blair, G. Coulson, N. Davies, "Standards and Platforms for Open Distributed Processing", *Eletronics & Communication Engineering Journal*, p.123–133, 1996.

[4] ISO/IEC 10746-1, *Information technology – Open Distributed Processing – Reference model: Overview*, 1st edition, 1998.

[5] E. Shahnam, "The Customer Relationship Management Ecosystem", *Delta Research Reports*, 2000.

[6] A. P. Gonçalves, "Proposta de Arquitetura Aberta de Central de Atendimento", Master's dissertation, *EPUSP*, 2001.

[7] J. D. Wells, W. L. Fuerst, J. Choobineh, "Managing Information Technology for One-to-one Customer Interaction", *Information & Management*, 1998.

[8] M. Feridum, G. D. Rodosek, "Management of IT Services", *Computer Networks*, v.43, pp 1-2, 2003.

[9] K. Geihs, "Middleware Challenges Ahead", *Computer*, pp 24-31, June 2001.

## Acknowledegements

The authors wish to thank the support of the UNILINS - Escola de Engenharia de Lins - Lins, Brazil.

# A Model-Driven Approach For Information System Migration

Raymonde Le Delliou[1], Nicolas Ploquin[2], Mariano Belaunde[3], Reda Bendraou[4], Louis Féraud[5]
*1 Electricité de France, 2 SOFT-MAINT, 3 France Télécom, 4 LIP6, 5 IRIT*
*1 juliette.le-delliou@edf.fr, 2 nploquin@sodifrance.fr, 3*
*mariano.belaunde@rd.francetelecom.com, 4 Reda.Bendraou@lip6.fr, 5 Louis.Feraud@irit.fr*

## Abstract

*In 2002 and 2003, the TRAMs project, led by a consortium of French companies and universities, experimented the application of emerging model engineering techniques to Information System (IS) migration. The main objectives were to define a methodology and to specify an open and modular migration framework to solve IS migration complexity on the basis of models and meta-models, as well as model transformation. A concrete result of this project was the development of a demonstrator that performed the migration of a large COBOL legacy application of an insurance company to use a JAVA and HTML based interface. In this migration use case, we used a common intermediate meta-model based on ISO/RM-ODP (Reference Model of Open Distributed Processing) extended with the UML Action Semantics formalism.*

***Keywords***—model driven architecture, modeling, meta-modeling, model transformation, Information System migration, legacy code.

## 1. INTRODUCTION

To deal with increasing competition, companies have to face constant strategic, organizational or technical changes which imply repetitive modifications of their Information Systems (IS). The massive introduction of internet technologies is one example. Though IS migration solutions exist on the market, all these solution are ad-doc solutions that are difficult to reuse and most of them rely on proprietary tools.

This paper will describe the experiments made in a cooperative French project called TRAMs to solve some of the challenges of IS modernization, by using extensively *model-oriented* engineering techniques. In particular there is an attempt to integrate RM-ODP with Action Semantics to enable fine-grained representation of the applications an enterprise may wish to migrate.

TRAMs, was launched in 2002 and was partially funded by the French government in the context of RNTL national research program. The consortium comprises:

- two French industrial groups, France Télécom and Electricité De France, having to migrate large legacy applications,
- two universities, LIP6 Paris 6 and IRIT Paul Sabatier Toulouse, known for their works on modeling, meta-modeling, languages and model transformation,
- SOFT-MAINT, a company specialized in large-scale IS migration projects.

One of the primary objectives of TRAMs's was the definition of a methodology and the provision of a generic architecture to help mastering IS migrations. In particular our ambition was to ensure that it is possible to reuse the good practices on migration technology in multiple and repetitive migration projects. The more efficiently an enterprise organizes knowledge reuse for their Information System (IS) evolution, the less it would cost at the end to integrate the new technologies, even if the initial investment may be high. One of the important issues also addressed by the project was the problem of maintenance after a migration occurs. Is maintenance facilitated when models are first class artifacts?

The approach taken by TRAMs to reduce IS migration complexity was to decompose a migration as a sequence of transformations that apply on models. A model represents an abstraction of an input, an output or an intermediate result. Some of these transformations may involve a lot of "manual" human operations – such as reverse engineering– while others may be totally automated. The inputs and outputs of IS

migration were re-formulated and formalized in terms of meta-models in order to apply model engineering techniques, like model-to-model transformation or code generation. In short, transforming an IS comes down to transforming models of the IS.

TRAMs makes use of various OMG modeling standards, such as MOF and UML for model representation [MOF] [UML], XMI for tool exchange [XMI], SPEM for migration process and process patterns description [SPEM] also In addition it used a combination of the ISO/RM-ODP standard [ISO95] [ISO2002] and UML's Action Semantics to define an intermediate common enterprise model capable of expressing low-level computations. The advantage of using these standards, instead of proprietary formalisms, is easy to understand if a company aims to take advantage of present or future products in the market place. However, our methodology and architecture does not enforce the usage of these standards as long as other formalisms are able to play the same role.

In the context of this work, we will assume the following definition of a model: a model is a description or a specification of a system and its environment for some certain purpose [MDAG]. In line with actual meta-modeling principles, a model will always be defined in the terms of a meta-model. For instance, a complete RM-ODP specification can be formalized as a collection of one or more models, each of them being defined in relation to specific meta-models, such as a meta-model for the enterprise view and another for the computational view.

## 2. METHODOLOGY AND FRAMEWORK OVERVIEW

In order to define a general methodology for information system migrations, TRAMs project has:

- Firstly, provided a typology of migration projects according to the kind of information that is impacted, For instance, we have migrations that imply changes on the business process, and/or in the applications interfaces and data. Some migrations are on technology oriented while other are fundamentally driven by a business change.
- Then, identified and implemented various process patterns, which may or may not include the usage of common intermediate formalism, which potentially can be reused in various migration projects.

  An important outcome of the project was the classification of migration activities that are part of the *preparation* from those that are part of the *execution* of the migration.

  In parallel to this work on methodology, the TRAMs project has specified an architecture – the so called TRAMs framework – that allows managing successive migrations based on the usage of model repositories and model transformers.

To define a migration framework one has to think about the relevant components needed:

- to prepare the migration,
- to execute the migration.

Preparing a Migration implies the ability to represent and store a description of the migration process to be performed. In the project, we used the SPEM formalism and notation and a MOF-based repository capable to store SPEM models.

Within the preparation phase, after designing the migration process model, one has to think how to implement each process activity. This was done by enriching the process description with the so-called "instrumentation model" which indicates what software components are used in each activity, or what kind of manual actions need to be done for the activity to be fulfilled. Storing a model of the implementation (the instrumentation model) ensures that information is not lost. This information can be re-used when a new migration has to be performed.

Migration preparation involves also the definition and/or the identification of the meta-models to represent the inputs or the outputs of the IS migration. Finally, each transformation component needs to be specified and implemented. In the case of a model-to-model transformation it is possible to use an executable specification language. Typically, a generic transformation engine will execute transformation specified as a list of rules. Section IV presents some of model transformation techniques experimented within this project.

The kinds of tools that are used during the execution of the migration are typically:

- Specialized reverse engineering tools – such and parsers and pattern analyzers – to scan the available legacy code and to discover any relevant high-level structure,
- CASE tools, with graphical capabilities, such as UML tools,

- Model repositories to store and publish the input, output or intermediate models,
- Model-to-model transformers and code generators.

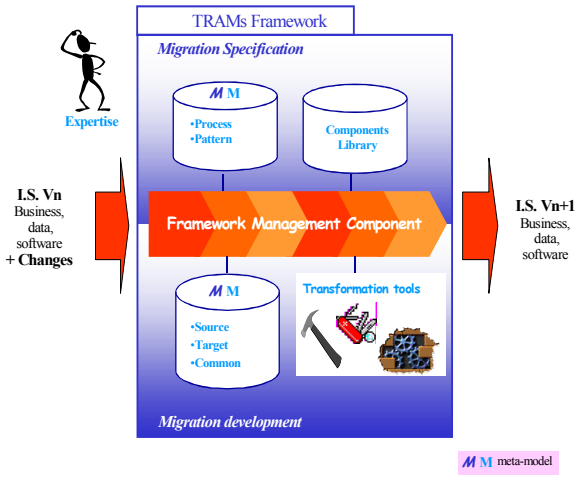



**Figure 1: TRAMs Framework**

The Figure 1 depicts the TRAMs framework architecture. On the top are showed the tools that are used for preparing the migration and in the bottom the tools used when executing the transformation. In the middle, a management component centralizes all information needed by a user to control and monitor the migration process.

To summarize, the TRAMs framework is **generic** and **open.** It does not impose any process or tool: each company can define its own migration process, use their own proprietary formalisms and connect their own tools. However, in order to take plain advantage of model-oriented engineering techniques it is important to use as much as possible data representations that are based on meta-models.

Meta-modeling is the key to achieve tool fine-grained inter-operability, as opposite of interface-based inter-operability that permits in most cases only coarse-grained inter-operability. For instance, any tool that supports SPEM models storage can play the role of the process repository as long as it supports import/export functions. In addition, any model transformer that knows the type of the input and output models and that implements the transformation specification rules can play the role of a transformer.

An initial implementation of the TRAMs framework principles was achieved by the end of 2002. This initial demonstrator performed the migration of the GUI capabilities of a large COBOL-based insurance application into two distinct targets: one based on HTML and Java script and a second one based on Java applets technology. The final demonstrator, presented in 2003 demonstrated the migration of business computational parts of the application. For that purpose an intermediate meta-model based on RM-ODP and Action semantics was used (see section III).

## 3. MODELING THE MIGRATION PROCESS

An example of a typical pattern for a migration project is depicted in Figure 2. This pattern results from the experience of the industrial partners involved in TRAMs..

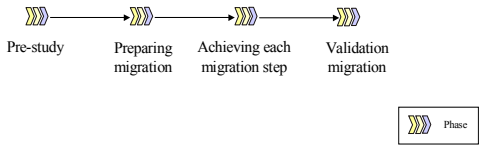



**Figure 2 : TRAMs process-type**

The pre-study is a phase in which the relevance for making a migration is done. Sometimes it is better to get rid of an obsolete application simply by re-written it completely from scratch. Sometimes, however, this is un-realistic, simply because no one is capable to understand the business rules that are hidden in an obsolete application!

In the "Preparing migration" phase, one has to:

- Identify the migration nature to measure migration complexity. Is it a business or technologic evolution? Does it imply changing the data storage, the user interfaces, the network elements, and so on.
- Identify – and if needed specify - the meta-models to be used during the migration. In the TRAMs 2003 final demonstrator, we used a COBOL meta-model in conjunction with a BMS meta-model to represent the source data. The BMS – *Basic Mapping Support* - is the transactional GUI system used in the insurance application. In addition, we used a meta-model to represent RM-ODP augmented with UML action semantics concepts.
- Model the migration process in SPEM, and try to reuse any pre-existing process pattern (if applicable). By modeling the process one may decide on the numbers of models that are to be managed separately – for instance the GUI aspect

may be put in a separated model in order not to pollute pure business data models. At this stage also, one may decide to use an intermediate meta-model that the company may want to use for various successive migrations.

The Figure 3 depicts the pattern that was used in the final demonstrator for the execution of the migration. . First, we have a reverse-engineering phase, then the input models are transformed – most automatically - in terms of the "neutral" intermediate meta-model. Then this intermediate representation is re-worked using heuristics and manual annotations into a more conceptual model. Finally this re-worked representation is used as the input for generating the artifacts needed by the target platform.

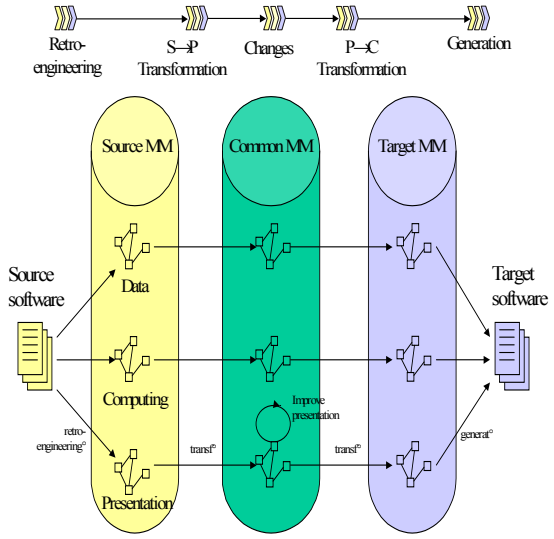



**Figure 3: Process pattern for the migration of a COBOL Interface to Java/HTML**

When a company needs to perform multiple and repetitive migrations, a common intermediate meta-model helps to:

- Decrease significantly the number of model transformers to be developed (for all the migrations to be done),
- Decrease times and costs of the information system evolution during further migrations. Thus, it may be possible to reuse the models stored in previous migrations as well as to reuse the transformers).

On the other hand a dedicated source-to-target solution may be much simpler to develop, simply because we can concentrate directly on the mapping rules that are needed in the specific migration case.

To summarize, the determining decision-making factors are:

- The number of similar migrations expected to be performed (or if it is a one-shot),
- The durability of the target applications, if they are to be maintained and the length of maintenance period,
- The availability of a transformer. In such a case, we can easily understand that we will tend to favor the option that reuse the available transformer rather than the option that requires developing new ones.

## 4. A COMMON INTERMEDIATE MIGRATION MODEL BASED ON RM-ODP AND ACTION SEMANTICS

The common intermediate meta-model used in the final TRAMs demonstrator is a collection of models based on ISO/RM-ODP (Reference Model of Open Distributed Processing) and on OMG's UML Action Semantics.

RM-ODP supplies the proper concepts for distributed computer system specifications. RM-ODP is based on an object approach. The system is described from five complementary viewpoints [IEEE-1471] [PUT], covering as well business aspects as the most technical aspects.

Identifying those viewpoints allows system specification to express at the same time but distinctly: the business the IS supports (Enterprise Viewpoint), the way it is modeled in the computer system regarding information and functions (Information Viewpoint, computational Viewpoint, Engineering Viewpoint) and the technical choices of the computer system mapping user requirements (Engineering Viewpoint, Technology Viewpoint).

The key points of RM-ODP are the sufficient completeness of its concepts and structuring rules and the relevance of its abstraction levels. In this way, the software architecture of the system to be built can be well specified using this set of concepts [BGL99] [BGL01] [BL01] [ODAC] [DASIBAO].

However, these abstractions are quite high level ones and do not allow to express the very detailed information that can be found in a software code. As an example, RM-ODP does not allow to specify actions connected with operations of a class nor does it allow specifying with enough detail their underlying

semantics. Migration applications clearly bring out the need for formalizing coding instructions, while remaining standard and platform-independent. Action Semantics turned out to be the right candidate for this criterion.

We then chose to integrate Action Semantics meta-model with RM-ODP meta-model.

We identified shared or common concepts from each standard, and displayed a logical continuity from one standard towards the other, i.e. from the more abstract concept towards the more detailed concept.

As an example, we will focus on the computational viewpoint, which is the central element of our two meta-models integration. Junction points were identified to achieve the coupling of this RM-ODP viewpoint with Action Semantics [SERP04].

If we consider the RM-ODP concepts of Action and Operation, according to the ODP standard, an Object has a Behavior, which is defined by a set of Actions and an Action is "something which happens".

In Action Semantics, the Action concept is a first class one, and is actually the fundamental concept of the Actions package. An Action is defined, in Action Semantics, as "a behavioral specification that acts on inputs to produce outputs". Even if both definitions do not strictly express the same semantic, it seems obvious that the ODP definition is more abstract and includes the one given by Action Semantics standard.

On the other hand, in ODP, an Interface is composed of Interactions, all of the same type (*Signal*, *Operation* or *Flow*). An Operation has a signature defined by a name as well as parameters. In UML, an *Operation* (defined in the Core package) corresponds to the specification of a *Method*. It has a name and zero or n parameters inherited respectively from the *ModelElement* and *BehavioralFeature*, abstract classes. A *Method* is implemented by a *Procedure* (package Actions/Action Foundation). Then we can easily map the ODP *Operation* concept to the UML *Operation*.

On Figure 4, the "Operation" rectangle represents both the UML and the ODP Operation concepts. This unique rectangle represents our first junction point and enables us to link the two meta-models. The second junction point is represented by the "Action" rectangle, which also links the two meta-models.

The final meta-model integrating ODP Computational viewpoint meta-model and UML Action Semantics meta-model (Figure 4) is now complete and covers both the software architecture and the coding instructions allowing a behavior specification, thus opening the way towards a complete code generation from models.

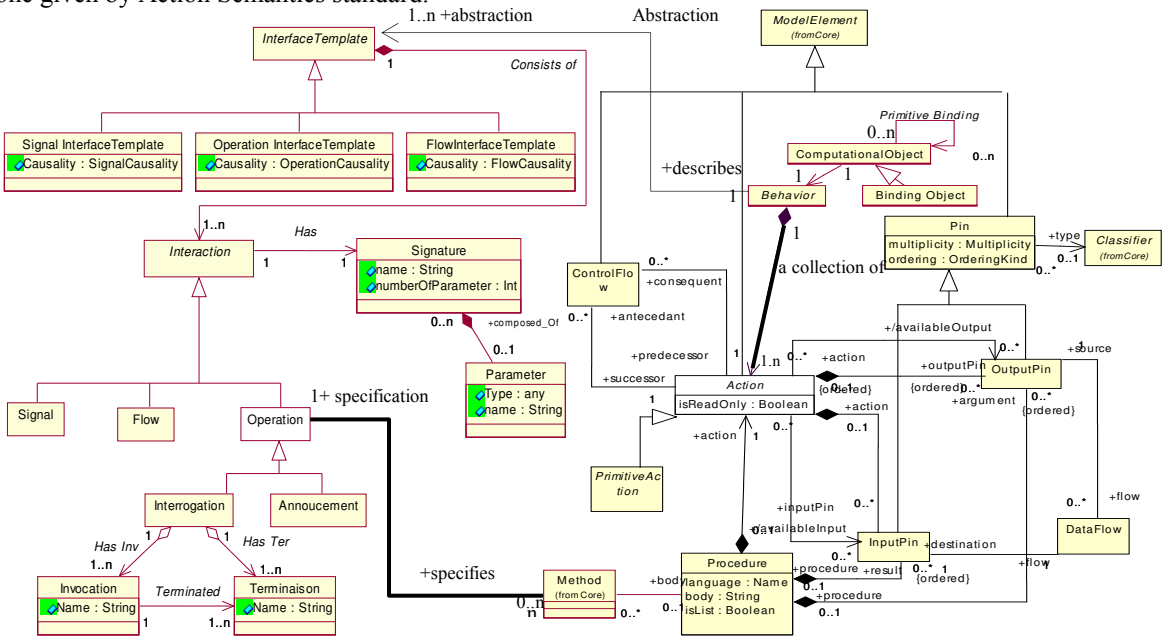



**Figure 4: An extract of the final meta-model coupling RM-ODP Computational viewpoint and UML Action Semantics**

## 5. USING MODEL TRANSFORMATION TECHNIQUES FOR TRANSLATING COBOL PROGRAMS INTO JAVA

The transformation process relies on model transformation techniques. TRAMs partners are involved in the OMG initiative for standardizing model transformation techniques known as MOF Q/V/T Request for Proposal. In this context, some of them already developed their own transformation engine.

TRAMs project thus has experimented three types of transformers, which are all based on rules expressing correspondence between concepts of two meta-models or within a meta-model:

- Model In Action (MIA), based on a SOFT-MAINT tool: the language is fully declarative and is based on predicate logic, ,
- TRL – formely called MTRANs - , based on a France Télécom tool: the rules have a declarative signature and an imperative body [BEL],
- Attribute Grammars: the rules are mathematic expressions in SSL [KAST] [KNUTH].

The rules are executed by transformation engines in compliance with the structure shown in Figure 5.

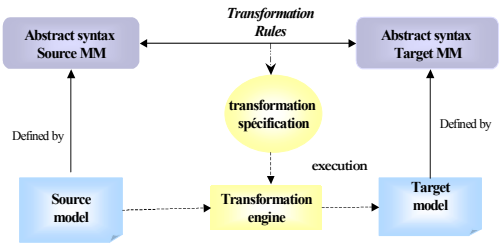



**Figure 5: Model transformation process**

All three approaches allow composing transformations, to trace concept evolution during transformation, to reuse the transformer or at least part of it. Only Attribute Grammars allow inferring modifications automatically, whereas MIA and TRL are the ones that can offer a user friendly concrete notation.

As shown in section II, a migration is typically divided in three sub-processes. As an example, we present here the transformation of computing information from COBOL into Java which was part of the 2003 demonstrator (Figure 6).


**Figure 6: Transformation sub-process for computing information**

This transformation is composed of several steps. The first consists in parsing source code from Cobol programs. This operation also consists in selecting relevant computing information. We extract control flow models by use of slicing techniques. Result models are based on a platform specific meta-model for Cobol source code, CobolSemTRAMs. This meta-model may be considered as an action meta-model and define concepts such as CobolProgram, CobolVariableData, CobolProcedure, If, While, Affectation, Comparison, etc.

The following step transform Cobol-oriented models into platform independent models based on the RM-ODP/Action Semantics meta-model presented in §III. Transformation rules are expressed by mapping COBOL meta-model concepts to Action Semantics meta-model ones.

Few mapping rules:

- CobolProgram → Class
- CobolVariableData → Attribute
- CobolProcedure → Method
- CobolStatementBlock → GroupAction
- If → ConditionalAction
- While → LoopAction
- Affectation → AddAttributeValueAction
- ArithmeticExpression → ApplyFunctionAction
- Comparison – ApplyFunctionAction

Action Semantics concepts are quite similar to Java actions concepts. Therefore, we decided to also consider the RM-ODP/Action Semantics meta-model as a target meta-model for this transformation sub-process. The last step thus consists in generating the Java source code from intermediate models. Each Cobol program is translated as a Java class.

The resulting program is merged with Java code generated by other transformation sub-processes. Final programs may be built and contain all required functionalities extracted from Cobol programs. However, the generated Java code stays excessively "Cobol-oriented". Transformation rules may be improved to get better Java programs.

## 6. CONCLUSION AND FURTHER WORKS

The TRAMs project has demonstrated how to take advantage of the emerging model-oriented techniques to facilitate successive migrations within a large company. As we mentioned in the introductory section, the cost of building a model-aware migration framework may be high, but return of investment can

be quickly be positive if the company has to perform various successive migrations to take advantage of the new technologies.

The TRAMs project final results were mainly a methodology and architectural principles that helps to apply the best practices for migration projects – like well-known migration patterns. The experiments achieved during this project have allowed SOFT-MAINT to improve their migration tools and to wide its offer towards other migration markets on which the company already has know-how stored in models (FORTRAN for example). We should note that TRAMs feedback in model transformation has also influenced the OpenQVT response (the so called "French submission") to the OMG's Q/V/T RFP.

This work has also made a fruitful attempt to combine the high-level architectural concepts of RM-ODP with low-level computational constructs bring by the Action Semantics standard. This integration has permitted us to use directly RM-ODP as an intermediate language for the representation of the applications that are part of the information system of an enterprise, and which could be subject to maintenance and evolution.

We believe that there is still a lot of research work that needs to be done in the field of information system migration. In particular, we need to study and classify the strategies that can improve the analysis of the code that is reversed as a model. The difficult part of a migration is still the ability to produce high-level models from low-level code.

## GLOSSARY

BMS: Basic Mapping Support
EDOC : Enterprise Distributed Object Computing
ISO : International Organization for Standardization
MDA : Model Driven Architecture
MOF : Meta Object Facility
OMG : Object Management Group
PIM Platform Independent Model
PSM Platform Specific Model
Q/V/T : Query/View/Transformation
RFP : Request For Proposal
RM-ODP : Reference Model of Open Distributed Processing
SPEM : Software Process Engineering Meta-model
UML : Unified Modeling Language
XMI : XML Metadata Interchange
XSLT : eXtensible Stylesheet Language Transformation
W3C : World Wide Web Consortium

## REFERENCES


**[BEL]** M. Belaunde, M. Peltier: From EDOC components to CCM components: a precise mapping specification (2002). *In Proc. ETAPS 2002*, Grenoble, France. LNCS 2306 Springer (2002)

**[BGL99]** X. Blanc, M.P. Gervais and R. Le Delliou, "Using the UML Language to Express the ODP Enterprise Concepts", in Proceedings of the 3rd International Enterprise Distributing Object Computing Conference (EDOC'99), IEEE Press (Ed), Mannheim, Germany, September 1999

**[BGL01]** X. Blanc, M-P. Gervais, R. Le Delliou "On the Construction of Distributed RM-ODP specifications" Proceeding of the Third IFIP International Working Conference on Distributed Applications and Interoperable Systems (DAIS 2001).

**[BL01]** X. Blanc, R. Le Delliou, "Information System architecture with RM-ODP: an on-the-field experience" Proceeding of the Open Distributed Processing: Enterprise, Computation, Knowledge, Engineering and Realisation (WOODPECKER 2001). pp27-37. June 2001.

**[DASIBAO]** A. Picault, P. Bedu, J. Le Delliou, J. Perrin, B. Traverson «Specifying an Information System architecture with DASIBAO, a standard based method", Proceeding of the International Conference on Enterprise Information Systems (ICEIS 2004)

**[IEEE-1471]** IEEE « Recommended practice for architectural description of software-intensive systems » IEEE Std 1471–2000.

**[ISO95]** ISO/IEC IS 10746-x, ITU-T Rec.X90x Open Distributed Processing-Reference Model Part x, 1995.

**[ISO2002]** ISO/IEC IS 15414 Open Distributed Processing Reference Model –Enterprise Language, may 2002

**[KAST]** U. Kastens: Ordered Attributed Grammars. *Acta Informatica* (1980), 13(3): 229-256

**[KNUTH]** D. Knuth: Semantics of context free languages. *Mathematical Systems theory* (1968)

**[MDA]** « Model Driven Architecture – Architecture board ORMSC » – document number ormsc/2001-07-01 – OMG-2001

**[MDAG]** « MDA Guide » – document number ab/2001-01-03 – OMG-2003

**[MOF]** OMG. "Meta-Object Facility (MOF) Specification v1.4". TC. Document formal/02-04-03 OMG. April 2002. http://www.omg.org

**[ODAC]** M.P. Gervais, ODAC : An Agent-Oriented Methodology Based on ODP Journal of Autonomous Agents and Multi-Agent Systems, Kluwer Publishers (Jan. 2002)

**[PUT]** J-R. Putman, « Architecting with RM-ODP », Prentice-Hal, 2001

**[SPEM]** Software Process Engineering Metamodel, Draft Adopted Specification, November 2001

**[SERP04]** R. Bendraou, S. Bouzitouna and M. P. Gervais, From MDA Platform-Specific Model to Code Generation: Coupling of RM-ODP and UML Action Semantics Standards, to appear in Proceedings of the International Conference on Software Engineering Research and Practice (SERP'04), Las Vegas, USA, June 2004

**[UML]** : "OMG Unified Modeling Language Specification", Object Management Group, March 2003, OMG TC Document UML1.5 (Action Semantics) formal/03-03-01, www.omg.org

**[XMI]** OMG "XML Metadata Interchange (XMI) v1.1". TC Document ad/99-10-02 OMG. 1999. http://www.omg.org

**[XML]** Extended Markup Language Version 1.0, W3C Recommandation
http://www.w3.org/TR/1998/REC-xml-19980210.pdf

# Challenges for ODP-based infrastructure for managing dynamic B2B networks

Lea Kutvonen
Department of Computer Science, University of Helsinki
Lea.Kutvonen@cs.Helsinki.FI

## Abstract

*The availability of open networks and the rise of service-oriented architectures have created an environment where collaboration between enterprise ICT systems becomes technically plausible. The current challenges for collaboration management focus on ensuring the semantics and pragmatics of collaborations. It is especially interesting to capture the inter-enterprise business processes in such a way that autonomous computing systems can control and manage collaborations of that form. Furthermore, management of open collaborations requires shared concepts and protocols for trust management.*

*The reference model of open distributed processing (RM-ODP) standards deal with distributed information processing systems that are exploited in a heterogeneous environment, under multiple organizational domains. The standards provide, besides general terminology and viewpoints for division of system specifications, a model for an infrastructure that supports distribution transparent communication at application level.*

*Our contributions to the field involve a B2B middleware architecture for managing application level collaborations in a evolvable and dynamic way. The work is consistent with RM-ODP, but extends the management and communication facilities. This paper discusses the challenges rising from this approach.*

## 1 Introduction

The globalization of business and commerce makes enterprises increasingly dependent on their cooperation partners; competition takes place between supply chains and networks of enterprises. In this competition, the flexibility of enterprise information systems becomes critical. The IT systems and development teams should be able to respond in a timely way to the requirements arising from the changing co-operation networks and their communications needs.

The availability of open networks and the rise of service-oriented architectures have created an environment where collaboration between enterprise ICT systems becomes technically feasible. The current challenges on collaboration management focus on ensuring the semantics and pragmatics of collaborations.

It is especially interesting to capture the inter-enterprise business processes in such a way that autonomous computing systems can control and manage collaborations of that form. We need automated processes - automated within reason and trust - for creating inter-enterprise relationships so that the selected business processes can span this new, temporary business network. In these processes, fundamental tools are those that ensure interoperability in a technically and semantically heterogeneous environment. On the other hand, we need environments where new business process models can be developed, published, and evolved. Business applications, business needs, and business network topologies change rapidly, and that change has to be reflected by new business process models. Even more frequently there are pressures on changing the membership of an existing business network - a company fails, another provides a better quality service, yet another has more robust suppliers.

This paper describes how the foundations of RM-ODP (the reference model of open distributed processing) [8, 9] have been expanded and interpreted in the web-Pilarcos project for the benefit of open business network management. The goal of the project is to develop middleware services that support inter-organizational co-operation. The web-Pilarcos project aims at managing dynamic communities in a way where membership requires technical, semantic and pragmatic interoperability. The architecture design specifically addresses the needs of independent evolution of computing platforms, application services, and operational policies in each enterprise involved. The solution gives special emphasis to runtime expression of pragmatic aspects. The perspective taken is of the interoperability middleware developer. The concepts and services of the interoperability middleware become available for applications, both for service providers and service users.

In addition, this paper discusses some of the challenges rising from the need for new, global infrastructure services for B2B collaboration management. Some of these chal-

lenges can be addressed by the ODP development community itself, some others need to be resolved through industry-driven consortia.

Section 2 outlines the required functionality for establishing eCommunities, controlling their behaviour, and changing their behaviour and structure during their lifetime, drawing attention to the challenges for B2B middleware. The required concepts and their relationship to RM-ODP concepts are discussed in Section 3. Section 4 briefly describes a B2B middleware architecture and services that are partially addressed by RM-ODP functions but have been enhanced and refined by our work. The paper is concluded by challenges for future work.

## 2 Composing and controlling eCommunities

Our essential goal is to support dynamic collaboration between service components (even, enterprise applications), across autonomic enterprises. The basic idea is very close to the ones behind virtual enterprises (VEs) or extended enterprises, and builds on loosely-coupled, autonomous services.

Traditional extended enterprise models evolved from the intra-organizational integration of enterprise applications. The B2B application integration solutions lead to tight coupling of applications based on data-oriented integration, application-interface oriented integration, method-oriented integration, portal-oriented integration or process-integration oriented integration [22]. On the other hand, ERP systems were burdened with heavy development cycle overhead, as enterprise application changes, IT computing platform changes, and business process changes were not supported. The next wave of systems took up a more dynamic approach [31]. The second phase ERP systems allow dynamic configurations of applications with peer-to-peer relationships. Also, the business process control aspects and integration of workflow management have strengthened the area. Furthermore, distributed business process management systems have started to emerge.

A move from static, monolithic extended enterprises to dynamically managed, loosely connected VEs has taken place. However, the connection between VE management mechanisms and business process management is not yet well developed.

The challenges met with the loosely-coupled, open collaboration networks are three-fold.

1. The participating enterprises should be autonomous, and furthermore, the services becoming part of the collaboration network should be autonomically administered.
2. The B2B middleware should provide automatic facilities for ensuring interoperability within the managed collaboration networks.
3. The B2B middleware environment should provide a set of concepts for managing collaboration network membership, conditions, and dynamics. These concepts should be supported by pervasive middleware services.

Autonomy is one of the key design aspects. It spans

- selection of computing platform, and schedule of technical changes in it,
- selection of service components put externally available,
- evolution life-cycle of each offered enterprise application, including withdrawal of services already part of some VEs,
- decisions on the kind of collaborations that are entered,
- decisions on the kind of partners are accepted, and
- decisions on leaving existing collaborations.

Within each collaboration, situations may rise where the operational goals of the collaboration and an enterprise contradict. In contradictory situations, enterprises should be autonomic in deciding whether they act according to their internal interests (and expect the sanctions of contract breaches) or comply with the VE rules.

The autonomy challenge is addressed by the use of service oriented architectures. For example, Web Services technologies provide a suitable frame for hiding technical processing differences. The engineering and deployment detail of service provision is left for the enterprises to manage, and between enterprises, only such service features are made visible (as metainformation) that are relevant to interoperability and managing dynamic changes in communication.

Another essential autonomy requirement of enterprises is that they should be able to determine the set of potential partner enterprises, a set of trusted partners. Trust information services should become one of the global infrastructure services (compare: DNS name service is a global infrastructure service). Trust should be tagged to each resource, client, and VE, and the levels of trust be dependent on the socially and technically correct behaviour of the element, as seen by others in the network. Trust management is an integral part of a VE architecture.

Collaboration between service components or enterprise applications require interoperability. Interoperability – i.e. the effective capability for mutual communication of information, proposals and commitments, requests and results – requires technical, semantic and pragmatic interoperability [5]. Technical interoperability means that messages can be transported from one application to another, for example using a common transport protocol, or other shared signaling method. Semantic interoperation means that the message content is understood in the same way by the senders

and the receivers. This may require transformations or other manipulation of messages, based on shared ontologies. Finally, pragmatic interoperability captures the willingness of partners for the actions necessary for the collaboration. The willingness to participate has two sides: capability of performing a requested action (for example, whether there is an application method available or not), and policy dictating whether the available action should or should not be performed (for example, a bank transfer handled after office hours).

The purpose of the B2B middleware is to provide a set of collaboration related concepts for application components to use, without the need of application software to include complex routines for example for partner discovery, interoperability guarantee, or change management. The concepts include community, role within the community, member of a community in a role, business process, policy, eCommunity contract, and contract breach, all of which have their counterparts in RM-ODP standards, as described in Section 3.

The most prevalent concept for our collaboration management architecture is the model of dynamic collaborations themselves, eCommunities. An eCommunity is controlled at operational time by an eCommunity contract. The eCommunity contract essentially uses a set of business process models as a model of the behaviour of members within the eCommunity. The collaborating services from each enterprise are aware of the business process model used between them, but do not implement the control of it. Instead, the control is left for B2B middleware services. These middleware services run metalevel protocols for controlling the eCommunity structure and state, including reports of contract breaches.

The life-cycle of an eCommunity has two modes:

- establishment phase supported by a breeding environment that ensures selection of appropriate partners and the interoperability of involved services, and
- operational phase supported by reflective control environment that manages dynamic changes in the eCommunity, and detects and resolves breaches of contracts.

The services of the breeding environment may be used even after reaching the operational phase, as the reflective control facilities may call upon restructuring of the eCommunity. The breeding environment should allow as open as possible route for enterprises to join in, giving an effective market for new partners. It provides facilities to a) populate an eCommunity to be created, b) negotiate eCommunity establishment, and c) commit eCommunity establishment.

The eCommunity management services at operational time are provided by the eCommunity contract object itself thought the following operations: a) terminate eCommunity, b) notify of entering compensation process, c) notify of detected eCommunity contract breach, d) query eCommunity contract metainformation and eCommunity status in terms of progress in the business process, membership, and breach management process definitions, e) repopulate and negotiate an existing eCommunity; and f) for members to join/leave an eCommunity role.

Access to these operations is made available at each platform at each administrative domain, regardless of whether the service is actually realized locally or remotely supported. (Different deployment models even open up new electronic service market opportunities.)

The middleware services providing for these services are discussed further in Section 4.

## 3 Refinement of RM-ODP concepts

The ODP standards provide for system architectures that allow distributed information processing applications to collaborate in a heterogeneous environment and under multiple organizational domains [7]. The ODP standards direct systems to be built so that they support cost-effective interoperability of applications, despite their implementation using different platform architectures and resources. For this, it is essential to accommodate system evolution and runtime changes, and define a transparency support framework for communication.

The RM-ODP provides a division of an ODP system specification into viewpoints, in order to simplify the description of complex systems [9]. The viewpoint languages each refine a set of general concepts defined for the reference model [8] The enterprise viewpoint language has been further developed [16], as it brings in business related aspects. Furthermore, the reference model defines structuring rules and functions for a supporting infrastructure for global computing. The functions that have been further standardized include the trading service [10], naming framework [12], type repository function [15], and interface binding framework [11] together with the supporting protocols [14].

In the web-Pilarcos architecture, we have tackled the challenges of autonomy, interoperability, and suitable concepts and services for managing eCommunities with these tools. Conceptually, the terms service and service offer management, community, federation and contract deserve further attention.

The concept of service is missing from the RM-ODP model, although it is mentioned in various term explanations (e.g. object [8, 8.1]). We define *service* as an abstract processing step that either creates, modifies, or consumes information, from the point of view of its environment. Services are made available at interfaces (seen from the computational or engineering viewpoints) and defined by the structural, behavioural and semantic rules of the interaction

involved (seen from the enterprise, information and computational viewpoints).

Services are provided by administrative domains, for example by enterprises, departments or any independent ICT systems. The administrative domains are the units of autonomy within our model. How engineering and deployment of services are organized within the administrative domain is hidden from the service users and B2B management services. Only management functions within the administrative domain, like node or object management, are involved with the technology and engineering of these.

In RM-ODP, models of behaviour (including interactions between objects and internal actions) are restricted to signals, announcements and interrogations, described using interface signatures. Signatures reveal operation names and data types, as well as parameters involved. However, for business processes, more complicated choreographies need to be expressed, although built on top of these basic primitives. In addition, these primitive actions need to be attached with nonfunctional features too, like QoS or trust.

For these challenges, enhancing RM-ODP with more elaborate conversation models is not the right solution. Instead, tools for introducing and reflecting such models are needed. Such models are specific to application areas, and may need to evolve in time, or may be negotiable between collaboration partners. In contrast to this, association of nonfunctional features to interfaces should become an integral part of the RM-ODP model, but again leaving the ontologies of features and usable values open for extention.

For the use in eCommunities, the services can be published by exporting *service offers* to a *trading service*. The service offer represents details of the behaviour of the service (what kind of application protocol needs to be followed using it), and other information further describing the nature of the service depending on the application domain.

The structuring rules of the web-Pilarcos style of service offers is captured in Figure 1. The structure is more detailed than can be found in the ODP trading function standard, which requires interface type name, interface reference, and some attributes as name-value pairs. What is to be noted here is that the service offer captures aspects from all five viewpoints, either as descriptions of the service to be provided or as a requirement to be fulfilled by the environment in the subsequent contract. This structure differs from OWL-S and UDDI based solutions (e.g. [30]) by not addressing the groundings (access details) or the actual location of services. These locating aspects are only captured by the eCommunity contract formed according the offers; the grounding aspects are considered private for each enterprise. Only the interoperability-related features are required.

By capturing all viewpoint aspects into the service offer, we address the interoperability ensuring challenge. Making metainformation available in such structured way, we use the interface matching mechanisms of the trading service to match all relevant aspects of interoperability at the same time. This of course applies only for static analysis; for example for policies subject to further changes, only dynamic monitoring can catch mismatches.

This kind of matching process requires that the service offers are expressed in commonly understood terms in the areas of business processes and services within that context, nonfunctional features associable to processes or services, and policy frameworks meaningful for the services in question. In this area, some ontology creation tools exist, but there is no consensus on what principle the ontologies should be organized on.

```
service offer         := ((interface syntax)
                          (interface protocol)
                          (information el format)
                          (nonfunct aspects)*)*
                          (policies)
                          (platform requirements)
                          (channel requirements)
interface syntax      := <IDL specification> |
                          <WSDL specification>
interface protocol    := <partial ordering
                          rules of operations
                          in syntax>
nonfunctional aspects:= <QoS offer> |
                          <trust requirement> |
                          <security mechanism name>
information element format
                      := <schema>
policies              := <policy framework
                          name> <policy name>
                          <policy value offer>
platform requirement := <platform name>
channel requirements := <channel type name>
                          <binding type name>
```

**Figure 1. Structure of service offers.**

The concept of *community* is used for describing the collaboration of several services. The ODP enterprise language specifies a community as a configuration of enterprise objects with a contract on their collective behaviour [16, 5.1.1]. The community specification includes [16, 5.2]

- a set of roles; the role specification gives requirements and restrictions for the behaviour of an object;
- rules for assigning enterprise objects to roles; the policy rules can address individual objects or relationships between objects, and can make restrictions on behavioural and non-behavioural properties of the potential objects;
- policies that apply to roles; policy values act as selectors for alternative behaviours for the objects – and thus also for the community;
- description of behaviour that changes the structure or the members of the community during its lifetime.

Each role [9, 9.14] in the community specification denotes a possible behaviour. The behaviour descriptions are

refined with policy statements indicating which parts of the behaviour are prohibited, permitted or obliged to take place and under what conditions.

A role can be populated by an object that represents another community. In this way, larger systems can be composed of subservices. Functional composition is better supported by inclusion of multiple community specifications into a system specification and definition of the relationships between communities.

A community specification may be divided into several epochs, each epoch [9, 10.5] presenting a different set of services supported by the community. For instance, a service might have a configuration phase and an operational phase; during the configuration phase only a management interface is available, but during the operational phase the actual service interfaces are also available.

The ODP community structure is used as a baseline for defining eCommunity contract structure in web-Pilarcos. The eCommunity contract has the structure that is outlined in Figure 2.

The eCommunity contract structure is determined by the selected business network model. This model is suggested by the initiator of the eCommunity establishment. The models need to be available through a shared repository, so all potential partners can assess whether the goal, structure and terms of the community are acceptable.

Besides roles to be populated by services, the template shows requirements for the binding objects that are needed for realizing interactions between roles. For bindings, we expect an explicit, open binding object [13, 1]. The binding object provides a framework and interface for the communication service. The binding object also provides a management interface through with the internal structures can be configured using the local communication facilities or additional helper components. The benefit of this model is that the requirements on shared platform services become minimal: We need common understanding of interface descriptions and common understanding of a few alternative communication channel structures.

In addition to the metainformation contents, the contract object provides operations for changing members and community structure.

For the purposes of web-Pilarcos architecture, we have connected the role behaviour to a service behaviour. Thus, assignment rules are directly related to import requests from the trading service for suitable service offers from potential members of the eCommunity. This is in line with current trend of service oriented architecture (SOA) where the definition of abstract service, service discovery, and semantic composition of services are topical [25, 24, 24].

Service oriented architecture, and more specifically, web services provide evidence for the industrial movement towards independently developed and administered services.

- *reference to the business network model;*
- *current epoch information;*
- *process for changing epoch;*
- *for each role*
  - *assignment rules that specify the requirements on*
    - *service type;*
    - *nonfunctional aspects;*
    - *restrictions on identity, participation on other eCommunities, etc;*
  - *conformance rules that are used for determining conformance to the role which the assigned component is in the role; similar as above;*
- *for each interaction relationship between roles*
  - *channel requirements*
  - *locations of the channel endpoints*
  - *QoS agreement*
  - *security agreement*
  - *information presentation formats*
- *for each policy that governs the choices between alternative behaviour patterns in the business network model*
  - *acceptable values or value ranges;*
- *references to alternative breach recovery processes;*
- *objective of the eCommunity (rules that can be used for various nondeterministic choices in the eCommunity; for example, what kind of attributes are more attractive when selecting a new member, available info for these rules is in service offers and in the business network model and in policy values of this eCommunity)*

**Figure 2. eCommunity contract information.**

The service itself has become the key element: the behaviour pattern, nonfunctional features, and contracting for providing a specified service in a context. The context is defined by an environment, including local and network resources, availability of required services, etc.

As the business network model captures services in terms of only their interface and external exchanges of information, the model is free from private information flows and workflows within the service providing organization or unit. This is a great benefit, as many industrial approaches on inter-enterprise workflow and business process modeling have reported that the current modeling languages and tools enforce too tight coupling between partners [4, 27]. The phenomenon is a natural consequence of the development history through integrated ERP systems, to A2A integra-

tion, and further to process-aware B2B integration. The VE approaches however do not yet have sufficient support for business process management.

The basic ODP concepts introduce communities and contracts as a way of expressing how the partners can reach a shared goal. However, no notation or further refinement has been given for expressing the goal or behaviour. We have chosen to use an ad-hoc enterprise viewpoint language for defining these aspects. The language resembles XML-based business process modeling languages and workflow languages; in practice, the service descriptions use enhanced WSDL descriptions [26].

We have not adopted the UML notations nor taken MDA [28, 3] as a driving force for the design. As discussed above, the focus in this work is not in the generation of service implementations themselves, but in composing eCommunities from existing services. We do not use the term reuse here, as the facilities for establishing collaborations is the primary goal. Naturally, the descriptions required by our infrastructure and those of MDA tools overlap, and this is to be considered as a great benefit. However, there is a difference between the bias towards ergonomic modeling tool view of the business network and the bias towards management software beneath.

It should be noted that with this work, we do not drive the standardization of domain specific business processes in itself, but standardization of facilities that help in evolution. A methodology where new standard processes or new suggestions can be published and adopted efficiently is a more persistent approach.

The ODP interface references and bindings [11] standard discusses management of *explicit binding objects*. Introduction of such binding objects with a set of selective transparency support and nonfunctional aspects management is needed. Many commercially interesting consortia recommendations on the area of inter-enterprise workflows, web services choreographies, and business process management systems expect a transaction-aware communication layer to appear. Although the current ODP standards create a placeholder for a unifying structure, it is not concrete enough to guide the isolated development trends for cross-platform protocols and services in this area.

The RM-ODP defines <*X*> *federation* as a community of <*X*> *domains* where there is a shared objective [9, 5.1.2,5.1.1]. A <X> domain is defined as a set of objects with a shared controlling object over the characteristic feature X [9, 10.3].

In the web-Pilarcos architecture, we form federations between eCommunity management agents in administrative domains. An administrative domain can be seen for example as an enterprise, a division, or another unit: essentially the domain is the unit of autonomy within our model. The objective of the federation of eCommunity management agents is collectively to form, maintain and use application level services from their domains in the roles of the eCommunity. As helpers, each agent has local management services, such as node or object management, monitors that are able to report breaches from the assumed interaction pattern, and binding factories. The shared goal of the management agent federation is to keep the agreed eCommunity running according to the contract.

## 4 Open B2B infrastructure services for collaboration management

In the web-Pilarcos environment middleware services for B2B collaboration management fall into two categories: cooperative management services for multidomain applications and local element management services [20].

The breeding environment services include only cooperative services:

- The standard trading service [23] for maintaining a repository of service offers, for the use of the enhanced trader.
- The enhanced version of ODP type repository [6] for holding relationship information between generic types (service types, binding types, interface types) that are technology-independent and used for matching purposes and technology-dependent templates that are used for instantiating the corresponding components and objects [17]. This mapping information is created by system programmers separately from business architecture descriptions and service offers.
- The repository for publishing and relating various business process models.
- The enhanced trader for populating business architectures with selected components [21]. The business process models contains roles as placeholders for services, but the selection of services for neighbouring roles is not independent, due to, for example, the need for shared binding requirements.
- The federation manager for negotiating, maintaining and renegotiating the eCommunity contract that represents an application instance.

The cooperative management services – enhanced trading, trading, type management, eCommunity management, federated binding – are all services that have a local server running in each domain. These active agents take care of making requests to their peers in other domains, as there is otherwise no authority to invoke management actions in a foreign domain [18]. The requests carry contracts to pass relevant meta-information that identifies what should be done and how.

The operational environment services include both local and cooperative services. The most essential cooperative service is that of the eCommunity contract object itself, with its management operations. The interface to the replicated contract object is maintained by all eCommunity managers.

In traditional protocol systems, interoperability could be verified statically, although with practical limitations and with expense. However, the process models described above cannot be fully verified any more. Two aspects, the deontic guards on actions and pragmatic monitoring of resources cause a situation where only some of the joint behaviour can be verified. For example, we can determine non-interoperability if available functionality is not sufficient for safe communication, leaving guards and pragmatic decisions aside. Or, non-interoperability can be stated if guards on actions are so contradictory that there are no acceptable traces through the process left.

Therefore, we need to add dynamic verification into the system, meaning introduction of

- monitors that enforce enterprise policies on resources (pragma) and notifies about discrepancies caused,
- monitoring of external service interaction conformance to the business process, and notification if inconsistent actions occur, and
- controllers of channels and notifications when channel properties have changed in a significant way.

The local service management adds lifecycle services and local bindings to this list.

- Service deployer for instantiating components for each role according to the contract that represents the application instance. The service deployer uses type repository information to map the contract onto appropriate technology solutions [21].
- Binding factory for instantiating communication channels between components. Because no remote instantiation service is supported across organizational boundaries, the binding factories at each computing system involved must cooperate. Again, the factories use repositories for mapping contract information onto appropriate engineering solutions [21, 19].
- Implementation repository for storing software packaging and maintaining their automatic installation scripts.

The overall view to the operational environment is twofold: First of all, the service components interact successfully with their peers in the eCommunity through binding objects. We call this the real system. Secondly, there is a level with a set of protocols for monitoring, configuring and reorganizing the real system constantly. We call this the metalevel. The relationship between these two layers is taken from reflective system design.

In reflective systems [2], the metainformation constantly describes the current real system structure, topology, state, qualities, etc. The essential part of the system infrastructure services are those facilities that are needed to keep the real system and the metadata in causal connection with each other. This means that changes in metainformation need to cause changes in the real system, and vice versa. For example, if an eCommunity member fails permanently, the eCommunity contract object reports that the member has abruptly left the eCommunity. Furthermore, the eCommunity contract object is proactive and starts the search for a replacing member in the eCommunity. After commitments from other members are received, the new member is joined to the eCommunity contract, and consequently, the service component is started up and bound to its peers through binding objects. Changes of metainformation can also be such that they only take effect later in the real system. For example, change in an enterprise policy is not necessarily effective immediately.

## 5 Conclusion

The above discussion shows that the RM-ODP concepts are suitable for modeling enhanced system software services for automated management of dynamic eCommunities. However, the standard definitions give fairly vague direction to the work.

In the design, we have noted the need for further ODP functions, such as

- business process model repository;
- community aware coordination functions; and
- trust management functions (build on top of security functions).

Interoperability support mechanisms require definition of new ontologies for defining terms and semantics for elements in the business or service models, and interoperability related attribute sets to be used together with service types. Similarly, ontologies for policy sets relevant for business processes and resources would be needed.

Furthermore, some existing concepts and framework standards need refinement. For example, concepts related to service and various alternatives for explicit binding object architectures were discussed above.

## 6 Acknowledgments

This article is based on work performed in the Pilarcos and web-Pilarcos projects at the Department of Computer Science at the University of Helsinki. The Pilarcos project was funded by the National Technology Agency TEKES in

Finland, Nokia, SysOpen and Tellabs. In web-Pilarcos, active partners have been VTT, Elisa and SysOpen. The web-Pilarcos project is a member in national ELO program (E-Business Logistics) [29]. The work is strongly integrated with RM-ODP standards work, and recently has found an interesting context in the FP6 INTEROP NoE collaboration.

## References


[1] G. S. Blair, G. Coulson, N. Davies, P. Robin, and T. Fritzpatric. Adaptive Middleware for Mobile Multimedia Applications. In *Proceedings of the 8th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 1997.

[2] G. Coulson. What is reflective middleware? *IEEE Distributed Systems Online*, 2003. Area on Reflective Middleware – http://dsonline.computer.org/middleware/RMaraticle1.htm.

[3] D. S. Frankel. *Model Driven Architecture - Applying MDA to Enterprise Computing*. OMG Press, 2003.

[4] D. Hollingsworth. *The Workflow Reference Model: 10 Years On*. Fujitsu Services, UK; Technical Committee Chair of WfMC, 2004.

[5] INTEROP NoE, EU FP6. Interoperability research for networked enterprises applications and software. http://interop.aquitaine-valley.fr/.

[6] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. ODP Type Repository Function*. IS14746.

[7] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. Part 1: Overview*, 1996. IS10746-1.

[8] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. Part 2: Foundations*, 1996. IS10746-2.

[9] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. Part 3: Architecture*, 1996. IS10746-3.

[10] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. ODP Trading function. Part 1: Specification*, 1997. IS13235-1.

[11] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing – ODP Interface References and Binding*, Jan. 1998. IS14753.

[12] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing – ODP Naming framework*, 1998. IS14771.

[13] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. Interface references and binding*, 1998. IS14753.

[14] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing – Protocol Support for Computational Interactions*, 1999.

[15] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. ODP Type repository function*, 1999. IS14746.

[16] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. ODP Enterprise Language*, 2003. IS13235.

[17] P. Kähkipuro, L. Marttinen, and L. Kutvonen. Reaching Interoperability through ODP type framework. In *TINA'96 Conference: The Convergence of Telecommunications and Distributed Computing Technologies*, pages 283 – 284. VDE Verlag, Aug. 1996. Extended abstract.

[18] L. Kutvonen. Management of Application Federations. In H. Konig, K. Geihs, and T. Preuss, editors, *International IFIP Working Conference on Distributed Applications and Interoperable Systems (DAIS'97)*, pages 33 – 46, Cottbus, Germany, Sept. 1997. Chapmann & Hall.

[19] L. Kutvonen. *Trading services in open distributed environments*. PhD thesis, Department of Computer Science, University of Helsinki, 1998.

[20] L. Kutvonen. Automated management of interorganisational applciations. In *EDOC2002*, 2002.

[21] L. Kutvonen, J. Haataja, E. Silfver, and M. Vähäaho. Pilarcos architecture. Technical report, Department of Computer Science, University of Helsinki, Mar. 2001. C-2001-10.

[22] D. S. Linthicum. *B2B Application Integration - eBusiness-Enable Your Enterprise*. 2001.

[23] Object Management Group. *OMG Trading Object Service Specification*, June 2000. OMG formal/2000-06-27.

[24] M. P. Papazoglou and D. Georgakopoulos. Service oriented computing. *Commun. ACM*, Oct. 2003.

[25] M. P. Papazouglou and W.-J. van den Heuvel. Service-oriented computing: State-of-the-art and open research issues.

[26] T. Ruokolainen. Component interoperability. Master's thesis, University of Helsinki, Department of Computer Science, 2004. In Finnish.

[27] K. Schulz, K.-D. Platte, T. Leidig, R. Guggaver, K. Elams, A. Zwegers, F. Lillehagen, G. Doumeingts, A. Berre, M. Anastasiou, M. Nunez, R. Goncalves, D. Chen, and M. Missikoff. A gap analyisis – interoperabilty development for enterprise application and software - road maps. Technical report, 2003.

[28] J. Siegel. *Developing in OMG's Model-Driven Architecture*. Object Management Group, Nov. 2001. White paper, revision 2.6.

[29] TEKES. *ELO program*, 2003. http://www.tekes.fi/programs/elo.

[30] The Intelligent Software Agents Group The Robotics Institute Carnegie Mellon University. Semanatic matchmaking for web services discovery. Technical report, 2003. http://www.damlsmm.ri.cmu.edu/.

[31] M. Ulieru and R. Unland. Emergent holonic enterprises: How to efficiently find and decide on good partners. *International Journal of Information Technology and Decision Making*, 2(4), Dec. 2003.

# Proposal for a Model Driven Approach to Creating a Tool to Support the RM-ODP

D.H.Akehurst
*University of Kent*
*D.H.Akehurst@kent.ac.uk*

## Abstract

*The potential for revising the RM-ODP standards is an excellent opportunity to move the given specifications into a form that is more amenable to the provision of tools that support the standard. Adoption of the approach advocated by the RM-ODP would be greatly increased if tools to support the approach were more readily available. This paper proposes an approach to generating such tools, directly, from a model based approach to specifying the RM-ODP viewpoint languages and correspondences. In particular this paper highlights certain requirements that the specification approach would need to meet if the production of such tools were to be achievable.*

## 1. Introduction

The Reference Model for Open Distributed Processing (RM-ODP) [11] belonging to the International Standards Organisation (ISO) was published about ten years ago. Although originally developed with respect to the telecommunications industry, as we move into a situation where more and more software systems are essentially distributed, this reference model becomes similarly more and more relevant to software development.

Over the last few years there has been more focus on supporting distribution and an increase in the number of different technologies to support distributed software systems; originally CORBA, COM, DCOM and lately Web Service based approaches such as .NET and J2EE.

Even though these distribution technologies have been proposed, the take up and use of the RM-ODP has not been as common place as its relevance and potential usefulness would lead us to expect. Contrast this with the outputs from the OMG, such as the UML and latterly their MDA approach, which appear to be widely used and are certainly commonly discussed.

There could be a number of reasons for this, marketing and publicity being one possible candidate. However, another reason is the accessibility of the OMG outputs which are nearly always supported by tools (even if they are not considered to be the best possible tools), these give practitioners a tangible artefact by which to evaluate and try out the proposed technologies.

Where are the easy to use graphical language based tools to support the RM-ODP? Perhaps such a tool cannot be created and at the same time be technically sound. However, it would be useful to have one that provides the easy accessibility offered by the numerous UML tools, which, it could be argued, are not always so technically sound but do provide an easy way into the world of OMG-Oriented Modelling.

Development of a simple graphical tool based on the RM-ODP is not quite as straight forward as it is for languages such as UML. Firstly, the RM-ODP proposes a five viewpoints approach to system design involving at least as many separate but related sets of concepts but specifically does not prescribe any particular language with which to write design specifications. Secondly, the concepts recommended for each viewpoint are described using English text (as opposed to the OMG's approach of modelling the language concepts – known as metamodelling).

Thus, before we can build a tool that supports the design of systems using the RM-ODP approach, we need to address these two issues. This position paper proposes using the OMG's approach to solving the second issue, by forming metamodels that define the concepts described in the ODP Standard documents, as has been done for some of the viewpoints already [2, 9, 10] The first issue, that of viewpoints and languages, we can address by firstly defining notations that map to the metamodel concepts and secondly by specifying the inter viewpoint relationships as relations between metamodel concepts. Both of these (inter viewpoint correspondences, and notations) can be defined using the relation based transformation specification technique taken from the MDA initiative [8].

In the following sections we first discus the five viewpoint language concepts and their corresponding metamodels; secondly we propose a technique for defining notations for each viewpoint; thirdly we illustrate an approach for specifying inter-viewpoint consistency relationships. The paper ends with a discussion about the

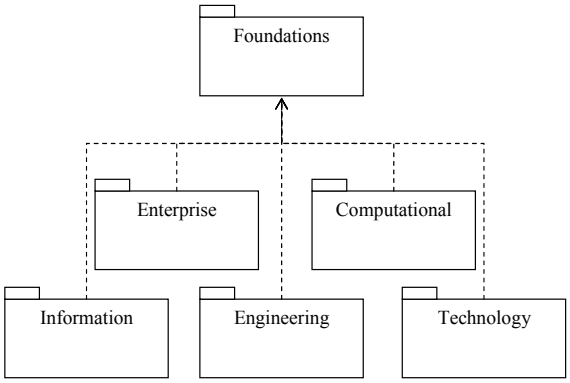



**Figure 1 Viewpoint Metamodels**

issues and problems to be addressed and a future direction for the work.

## 2. Viewpoint Metamodels

A number of papers have proposed metamodels for different RM-ODP viewpoints; [10] proposes an Enterprise Viewpoint metamodel; [9] and [2] discus Computational Viewpoint metamodels; and [7] describes a metamodel for the RM-ODP Foundation concepts.

A RM-ODP tool could be based on a set of five such metamodels, with the addition of (at least) a common 'Foundations' metamodel. Using an OMG MOF like language we can illustrate the approach as shown in Figures 1-3. Figure 1 shows an overview of six packages containing the metamodels for the five viewpoints and a metamodel for the common Foundations. Figure 2 and Figure 3 show example segments of possible Computational and Engineering Viewpoint metamodels.

The specification of viewpoint (or language) concepts in this manner makes it very easy to generate core parts of tools that support the viewpoints. Tools such as the Eclipse Modelling Framework (EMF) [5] and Kent Modelling Frameworks (KMF) [6] easily generate basic repositories that can form the core of a design tool from such models of a language.

## 3. Viewpoint Notations

In addition to repository functions a useful design tool requires a mechanism for populating the repository using

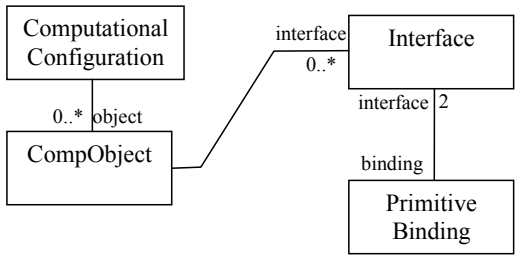



**Figure 2 Computational Objects**

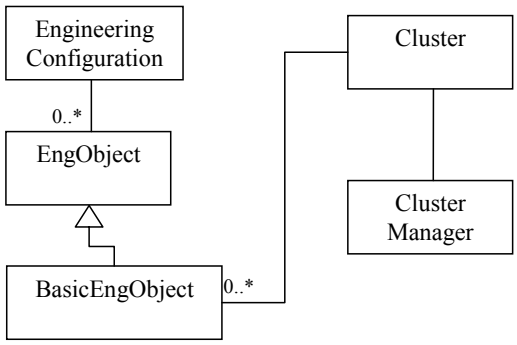



**Figure 3 Engineering Objects**

appropriate notations (or concrete syntax). The ODP ISO standards along with books such as [Blair/Stefani] make use of notations for describing various viewpoint designs. However although clearly understandable the notations are informally defined. If we can define these notations more formally, then there is scope for auto-generation of tools to support the viewpoint languages.

In [1] and [4] a modelling approach to defining visual languages is described. The approach is to define the concrete syntax of the notation as a model and subsequently specify a model transformation (via a set of relations) between the concrete syntax model and the concepts metamodel.

Using notation from the latest submission to the OMG's QVT RFP [8] and the technique defined in [3], an example for part of a Computational Viewpoint language specification is illustrated in Figure 4. It shows the specification of relations between Computational Objects and Circles, and between Interfaces and Solid rectangles (or bars); an example of the notation is shown in Figure 5.

The detail of the relations must be added to the graphical view of them, defining the domain and range of the relation and a matching condition expression that specifies which elements from one side (domain or range) of the relation are mapped to elements from the other. It is also necessary to define characteristics of the relation such as whether it is functional, total, bijective, etc. (The detail of the relations is shown in an Appendix.)

This approach can be used to define notations for all of the viewpoint concepts giving us a set of models and

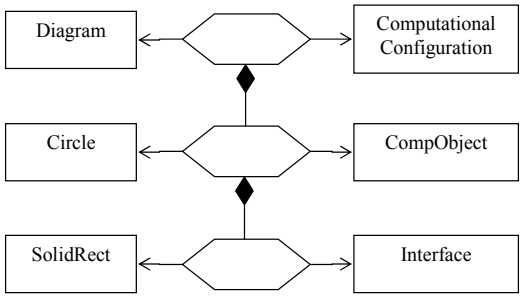



**Figure 4 Syntax-CompVP Relations**

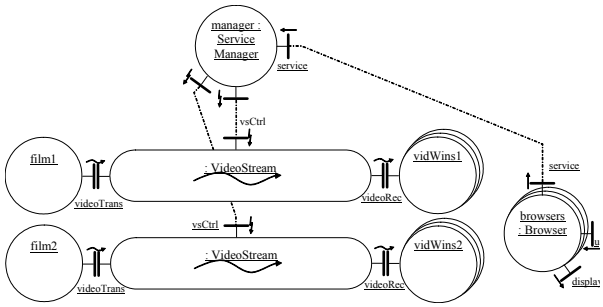



**Figure 5 A Computational Configuration**

relations from which it is possible to generate a tool that facilitates drawing designs in each of the viewpoint languages, including the necessary features of such a tool offered by a model repository.

The KMF tool developed at the University of Kent [6] has been used to generate parts of such tools for visual languages from this type of specification.

## 4. Inter-Viewpoint Consistency

A key part of the RM-ODP is the inter-viewpoint consistency specifications that tie the information from the five viewpoints into a consistent design from which an implementation can be produced.

Similarly to the specifications linking concrete syntax to metamodel concepts, we can define relations that link concepts between the different viewpoints. However, for some of the consistency relationships it is not possible to define them at the meta-level. It is necessary for the connections to be made at the design stage, for this we need another language (or at least a tool mechanism).

Figure 6 shows inter viewpoint consistency relations between concepts from the Computational Viewpoint and the Engineering viewpoint. This relations model the following correspondence:

"Each computational object which is not a binding object corresponds to a set of one or more basic engineering objects (and any channels which connect them). All the basic engineering objects in the set correspond only to that computational object."

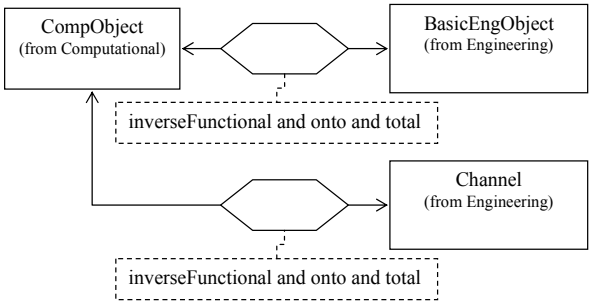



**Figure 6 A Comp-Eng Correspondence**

The relations are characterised as 'inverseFunctional' (or 1-to-many), each 'single' CompObject maps to 'many' BasicEngObjects (or Channels), but each BasicEngObject (or Channel) maps to a single CompObject. The relations are also defined as 'onto' and 'total' specifying that every CompObject and BasicEngObject (or Channel) in the domain and range of the relations must be part of the relation.

It is not feasible to define a matching condition for these relations (i.e. an expression that would define which objects from domain and range are mapped to each other) because there is no way at the meta-level to determine which objects should be related. The correspondences must be set up by the designer on a per design basis; however, the specification of these relations will enable a generated tool to indicate whether on not the correspondences have been set up.

## 5. Conclusion, Issues and Future Work

The previous sections have given an idea of how model based specifications of language concepts and relations between them can be given to define aspects from the RM-ODP standard. Using code generation techniques these types of specification can be used to generate tools that support designing a system from the different ODP viewpoints.

This paper has illustrated the ideas using languages provided by the OMG However, OMG languages are not essential, any precise approach to defining the languages and relationships would provide the necessary starting point for generating tools. In particular the following specifications should be given:

1. Precise definitions of the viewpoint language concepts – beyond the textual descriptions currently given – i.e. metamodels.
2. Precise specification of the correspondences between concepts in each viewpoint, along with suggested mechanisms for specifying these – i.e. a Correspondence Specification Viewpoint.
3. Precise specification of *example* notations for each viewpoint. Perhaps both graphical and textual.

In addition, a number of **full** example system designs should be provided illustrating intended use of the framework.

The approach to generating tools presented in the paper is an initial idea, a number of issues and problems are likely to make it difficult. Some of these are discussed below:

- Metamodelling – What concepts should be used to define the metamodels? Sections 6 and 7 of Part 2 of the standard informally define some language definition concepts, is it possible (or even a good idea!) to extend these to give a full and sufficient metamodelling language.

- Relations – Is the proposed technique for specifying relations expressive enough for the proposed task. Some correspondences are not at all easy to define! How useful or possible is it to define them all as relations?
- Correspondences - we need a mechanism for a designer to specify correspondences, i.e. we need a viewpoint for defining inter-viewpoint correspondences.
- Technology Viewpoint – the concepts for a metamodel of this is not obvious, in fact the concepts currently in the standard are minimal. Is it necessary to have something more and if so what should be in it?

Extending the idea of a supporting tool; it would be very useful to provide MDA like support for generation of system implementations (or simulation/analysis models) from the given designs. To facilitate this we certainly need good definitions of Technology models.

Generating an implementation could be achieved using an MDA like approach of transforming information drawn from designs given in the five viewpoints and providing a set of implementation source code, configuration, and deployment files.

## 6. References


[1] Akehurst D. H., "An OO Visual Language Definition Approach Supporting Multiple Views," in proceedings VL2000, IEEE Symposium on Visual Languages, September 2000.

[2] Akehurst D. H., Derrick J., and Waters A. G., "Addressing Computational Viewpoint Design," in proceedings Enterprise Distributed Object Computing Conference, EDOC 2003, Brisbane, Australia, pp. 147, September 2003.

[3] Akehurst D. H., Kent S., and Patrascoiu O., "A relational approach to defining and implementing transformations between metamodels," *Journal on Software and Systems Modeling*, vol. 2, pp. 215, November 2003.

[4] Clark A., Evans A., and Kent S., "Engineering modelling languages: A precise meta-modelling approach," in proceedings ETAPS 02 FASE Conference, LNCS, Springer, April 2002.

[5] IBM, "Eclipse Modeling Framework," http://www.eclipse.org/emf/, 2003.

[6] KMF-team, "Kent Modelling Framework (KMF)," 2002, www.cs.kent.ac.uk/projects/kmf

[7] Naumenko A., Wegmann A., Genilloud G., and Frank W. F., "Proposal for a formal foundation of RM-ODP concepts," in proceedings ICEIS 2001, Workshop On Open Distributed Processing - WOODPECKER`2001, Setúbal, Portugal, pp. 81-97, July 2001.

[8] OMG, "Request for Proposal: MOF 2.0 Query / Views / Transformations RFP," Object Management Group, ad/2002-04-10, April 2002.

[9] Romero R. and Vallecillo A., "Formalizing ODP Computational Viewpoint Specifications in Maude," in proceedings EDOC 2004, Monterey, California, September 2004.

[10] Steen M. and Derrick J., "ODP Enterprise Viewpoint Specification," *Computer Standards and Interfaces*, vol. 22, pp. 165-189, September 2000.

[11] X.901-5, "Information Technology - Open Distributed Processing - Reference Model: All Parts," ITU-T Recommendation, 1996-99.


## Appendix A

The detail of a relation specification needs to define a matching condition that indicates which elements of domain and range should be related. In addition the links between relations should define the contents of the domain and range sets for the sub relations. The following specifications indicate the approach for the relations given in Figure 4. Depending on the level of difference there is between the related components, the complexity of the expressions in the relation specification will vary.

```
relation {
  domainType : Diagram
  rangeType : ComputationalConfiguration
  matchCond : true
  subRel : { CircleRelCompObj(
                  diagram.circles,
                  config.objects ) }
}

relation {
  domainType : Circle
  rangeType : CompObject
  matchCond : compObj.name = circle.label.text
  subRel : { RectangleRelInterface(
                  circle.connections,
                  object.interfaces ) }
}
```