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.
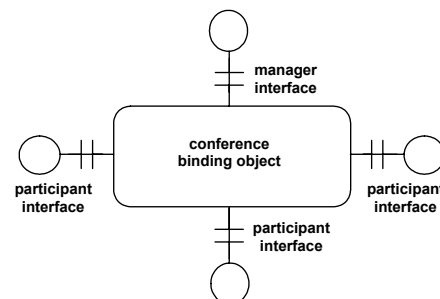
Figure 1 Snapshot of conference application

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.
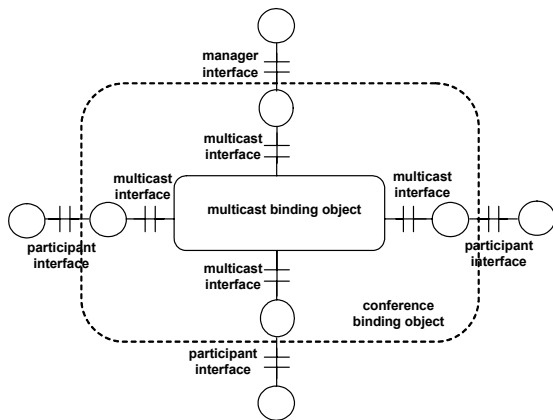

Figure 2 Revealing binding decomposition

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.
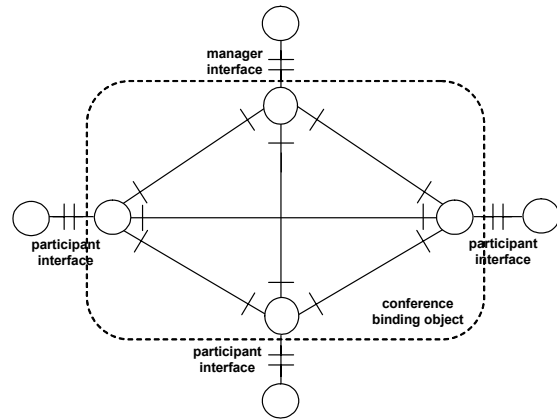

Figure 3 Revealing binding decomposition

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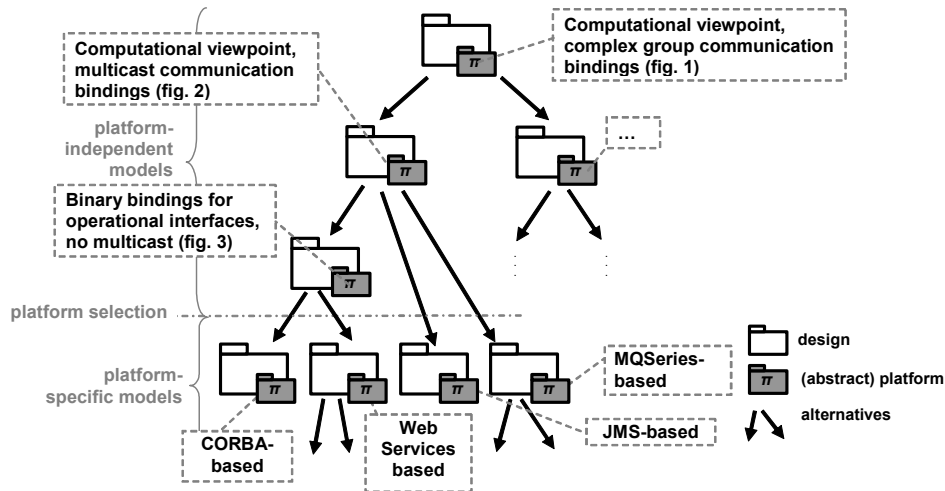

Figure 4 Models at related levels of platform-independence

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.
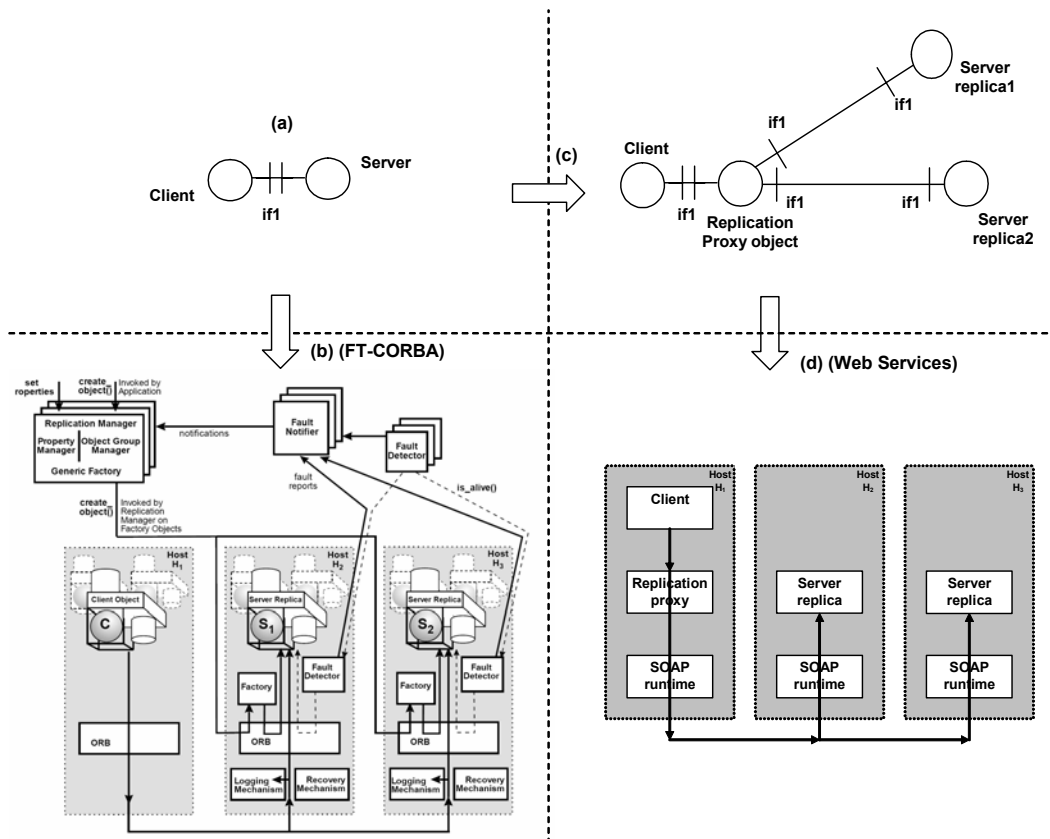

Figure 5 Alternative mappings for abstract platform with replication transparency

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

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