Model-driven Design of Distributed Applications

João Paulo A. Almeida

Centre for Telematics and Information Technology, University of Twente
PO Box 217, 7500 AE Enschede, The Netherlands
almeida@cs.utwente.nl

Abstract. The objective of the Ph.D. work discussed in this paper is to define a methodology for the design of distributed applications, in line with the Model-Driven Architecture (MDA). An important characteristic of this methodology is that it leads to models of distributed applications that withstand the impact of change in (middleware) platform technologies. These models are organized into different levels of platform-independence that are defined using the notion of abstract platform. An abstract platform is an abstraction of infrastructure characteristics assumed for models of an application at some point of (the platform-independent phase of) the design process. We aim at providing methodological guidelines for the definition of abstract platforms and their representations in modelling languages.

1 Introduction

The timely development of distributed applications is a costly effort. Therefore, an important quality of these applications is their ability to withstand the impact of change, both with respect to changes in application requirements and with respect to changes in the technologies used to build the application.

In the last decades, the development of distributed applications has been facilitated to some extent by the introduction of middleware platforms (e.g., CORBA/CCM [20], .NET [17], JMS [29], and Web Services [30, 31]). These platforms offer generic (distribution) support for distributed applications, masking from applications some details and differences in the support offered by programming languages, operating systems and network protocols. Since a significant amount of development effort is spent on overcoming problems related to distribution (e.g., remoteness, partial failures, heterogeneity) and in exploiting distribution beneficially (e.g., to achieve performance and dependability), the reuse of middleware platforms significantly increases the efficiency of application development.

Different middleware platforms have been developed, to satisfy a variety of needs. The current distributed application scenario is populated by multiple platform standards, implementations from different vendors, proprietary platforms and ad hoc infrastructures, standard and proprietary extensions to platforms, etc. These infrastructures provide different constructs from which applications can be built, and exhibit different quality characteristics. Recently, it has become clear that different parts of a distributed application may be built using various middleware platforms, and that the set of platforms used may change over time. In addition, it has also

become clear that middleware platforms may evolve during the lifetime of applications. The use of a single immutable distribution infrastructure is therefore not envisioned as a long term solution for the support of distributed applications.

The Object Management Group (OMG) has identified the need to address some of these issues in its Model Driven Architecture (MDA) [19], [21]. This architecture proposes the separation of platform-independent and platform-specific aspects of distributed applications into platform-independent models (PIMs) and platform-specific models (PSMs). A common pattern of MDA development is to define a platform-independent model (PIM) of an application, and to apply (parameterised) transformations to this PIM to obtain one or more platform-specific models (PSMs).

The potential benefits of this approach stem from the possibility to derive different PSMs from the same PIM, and to partially automate the model transformation process and the realization of the distributed application on specific target platforms. While this may reduce development costs and improve software quality, it also forms the basis for facilitating the evolution of software solutions, hence contributing to the containment of maintenance costs for distributed applications.

Nevertheless, the appropriateness of MDA as an approach for the development of distributed applications can be criticized on a number of points:

- there is a lack of guidelines to select abstraction criteria and modelling concepts for platform-independent models;
- there is little methodological support to distinguish between platform-independent and platform-specific concerns, which is detrimental to the beneficial exploitation of the PIM-PSM separation of concerns;
- the distinction between platform-independent and platform-specific models is coarse and insufficient to cope with the diversity of application requirements and infrastructure characteristics;
- little attention is given to the role of platform characteristics throughout the development trajectory, possibly leading to models with unacceptable levels of platform-independence and applications with unacceptable quality attributes;
- design operations are not clearly defined, thus inhibiting their effective application along a design trajectory; and
- the focus on a particular design language (UML) constrains the designer in some respects. Currently, it is unclear when and where such constraints apply.

In order to obtain the potential benefits of the model-driven approach to the development of distributed applications, we aim at addressing the issues above in an effective model-driven design methodology. The objective of our work is to propose such a methodology for the design of distributed applications so that:

- available and future distribution infrastructures can be (re-)used, improving the efficiency of the design process;
- the knowledge used to perform various design operations can be captured and re-used to improve the overall efficiency of the design process;
- designs of distributed applications remain stable in face of changes in platform technologies; and,
- designs can be reused to target different middleware platforms.

The methodology is defined so as to be generic with respect to application domains and platform characteristics.

This paper is further structured as follows: section 2 introduces the role of models and discusses the concept of abstract platform; section 3 discusses important activities of the proposed methodology, including abstract platform definition, abstract platform representation and transformation definition; section 4 discusses some work-in-progress; section 5 reviews related work; and, finally, section 6 presents concluding remarks.

2 The Role of Models

2.1 Platform-Independent Models

The development of a distributed application can be regarded as the process of building a realization of the application that satisfies user requirements. In most traditional development cultures, application developers are instructed to produce intermediate models to facilitate bridging the gap between requirements and realization. These intermediate models are mainly regarded as a means to obtain a realization of the system, with different models addressing different design concerns. The ultimate product of the development process is the realization, which can be deployed on available implementation technologies (platforms). Any intermediate models produced during the development processes are considered means and not ends.

In the case of Model-Driven Architecture (MDA) development [21], however, intermediate models that are used to produce the final realization are also considered final products of the development process. These models are carefully defined so as to remain stable in face of changes in platform technologies, and are therefore called platform-independent models (PIMs).

A platform-independent model can be refined or implemented into a number of technology platforms. For the purpose of our work, we assume that a platform corresponds ultimately to some specific middleware technology, such as CORBA/CCM [20], .NET [17], and Web Services [30, 31].

When pursuing platform-independence, 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 [7] and RM-ODP Enterprise Viewpoint models [13] (which can be considered Computation Independent Models [21]). However, when the application is described as a decomposition of interacting application parts, different sets of modelling concepts may be used, each of which is better suited for specific classes of target middleware platforms. For example, a designer may describe the interaction between application parts using either request-response invocations or event queues.

The implicit assumption of platform characteristics may result in models that cannot be reused for different platforms. Furthermore, it may lead to models of different applications that cannot be directly compared and integrated. We conclude that platform characteristics assumed in platform-independent models are better

understood and controlled by designers if they are explicitly represented. In our design approach, these characteristics are embodied in an abstract platform.

2.2 Abstract Platforms

The notion of abstract platform, as we have proposed initially in [2], supports a designer in defining levels of platform-independence explicitly. An abstract platform is an abstraction of infrastructure characteristics assumed in the construction of PIMs of an application. Alternatively, an abstract platform defines characteristics that must have proper mappings onto the set of concrete target platforms that are considered for a design.

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.

The use of the abstract platform concept may be reflected in an abstract platform model, as depicted in the in Figure 1. The PIM of a distributed application depends on an abstract platform model, in the same way as the PSM depends on a (concrete) platform model.

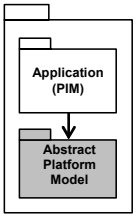

Fig. 1. PIM depends on abstract platform model

3 Methodological Aspects of Model-Driven Design

3.1 Abstract Platform Definition

The number of levels of platform-independence and the characteristics of the models at each level depend on a number of design goals to be balanced, including those of maximizing the efficiency of the design process and maximizing the reusability of models. Different application domain requirements and platform characteristics may also lead to the definition of different levels of platform-independence. We propose the levels of platform-independence should be identified explicitly in an early stage of the design process, which we call *preparation phase* [10]. In the preparation phase, (MDA) experts define the required levels of models as well as define the modelling language(s) to be used in the *execution phase*.

The definition of an abstract platform is supported by two observations [3]:

1. *platform characteristics may play a role in early (platform-independent) designs,* and;
2. *platform-independence must be balanced against platform-specific realization*

The first observation leads us to the conclusion that platform characteristics that play a role in platform-independent designs should be reflected in the abstract platform.

The second observation recognizes that achieving platform-independence is a requirement that must be considered in a larger context, where other relevant design goals play an important role. An MDA design process should lead efficiently to a (platform-specific) application running on a concrete platform.

The next subsections examine these observations and their implications.

Role of Platform Characteristics

Defining an abstract platform requires the ability to identify what abstract platform characteristics are relevant at a platform-independent level. Some platform characteristics become relevant when identifying application parts and their interactions. This is the case for the characteristics of the support for interactions between system parts. Some other platform characteristics play a more subtle, but not necessarily negligible, role. Platform characteristics that may have impact in early stages of the definition of a distributed application's architecture are likely to qualify as abstract platform characteristics.

This is best illustrated by an example, in which the design of a groupware service is considered. This service facilitates the interaction of users residing in different hosts. Initially, the service designer describes the groupware service solely from its external perspective, possibly stating quality-of-service requirements on the service, e.g., that the service should have high availability. At subsequent stages of development, the designer is confronted with design decisions. In this example, we consider the following alternatives: (i) a centralized (server-based) design, and (ii) a distributed (peer-to-peer) design.

Figure 2 depicts these two solutions. In solution (i), a server facilitates the interaction between users. In solution (ii), symmetric components facilitate the interaction without the support of a centralized application-level component.

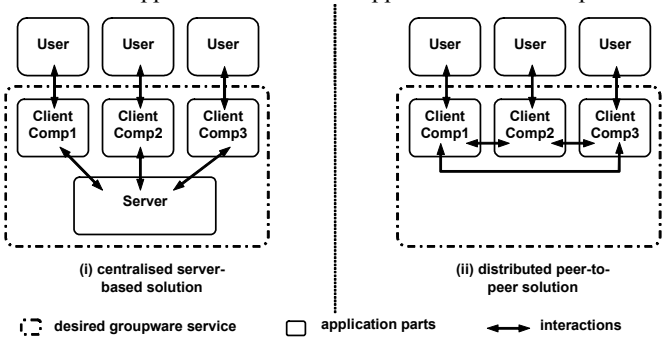

Fig. 2. Alternative designs for the groupware service

In order to improve the reusability of platform-independent models, stable aspects of a system's architecture should be captured in platform-independent models. Therefore, it would be desirable to select between alternative models (i) and (ii) during platform-independent modelling. Nevertheless, some platform-specific aspects play an important role in the selection of an adequate architecture. For example, solution (i) would introduce a single point of failure in the architecture, unless the platform provides support for replication transparency (as defined in the Reference Model for Open Distributed Processing (RM-ODP) [12]).

Apparently, this places the designer in a dilemma, since platform selection would affect platform-independent design. In order to solve this, a designer should be able to express, at a platform-independent level, requirements on platform-specific realizations that would allow all design decisions that are relevant for platform-independent modelling to be captured. In our groupware service example, this would mean that requirements on the reliability of individual components should be stated at the platform-independent level, justifying the selection of a centralized or a distributed design (possibly through application of aspect-oriented modelling [11]).

Requirements expressed at a platform-independent level should justify design decisions for the design at that level and provide input for platform-specific realization. If these requirements invalidate portability requirements for platform-independent designs, then it is impossible to consider the design at the current level of platform-independence. In this case, we envision two different contrasting solutions:

1. to consider the design at a higher level of abstraction, at which the platform characteristics are no longer relevant for design decisions taken at that level; or,
2. to relax portability requirements, lowering the degree of platform-independence for the design. This solution reflects on the characteristics of the abstract platform being defined.

For our groupware service example, possible applications of these solutions would be:

1. to describe the groupware service solely from its external perspective. At this level of abstraction, the reliability characteristics of the supporting infrastructure are irrelevant. Details on the service's internal design are only addressed at platform-specific modelling, and hence cannot be re-used for different target platforms; and,
2. to restrict the set of potential target platforms, e.g., to include only platforms that provide support for highly available components. In this case, it is possible to describe the groupware service's internal design at the newly defined level of platform-independence, while still guaranteeing the satisfaction of the service requirements. The abstract platform considered provides support for highly available components.

In [6], we have presented thoroughly an example of solution (2), where an abstract platform that supports dynamic reconfiguration of components is used at some point of the design process in order to satisfy availability requirements.

Platform-independence Balanced with Platform-Specific Realization

Defining an abstract platform brings attention to *balancing* between two conflicting goals: (i) platform-independent modelling, and (ii) platform-specific realization. On the one hand, an abstract platform indicates directly the support available for designers during platform-independent modelling, and therefore, reflects the needs of application designers, including the needs to handle complexity in application design

and portability requirements. On the other hand, an abstract platform is established by considering the set of potential target platforms and their (common and diverging) characteristics [2]; this bottom-up knowledge is useful to reduce the design space to be explored for platform-specific realization. Large design spaces are less amenable to automatic exploration, and require more intervention of designer, e.g., through extensive parameterization of transformations. Reducing the design space contributes to increasing the efficiency of the design process.

3.2 Abstract Platform Representation

Designs must be represented using suitable design languages. In a model-driven design process, several design languages may be used, e.g., to produce models at different levels of abstraction and platform-independence. Alternatively, a single "broad spectrum" design language [9] may be used. The design language adopted for a design has an important role in defining characteristics of an abstract platform assumed for the design.

In the *implicit abstract platform definition* approach characteristics of an abstract platform are implied by the set of design concepts used for describing the platform-independent model of a distributed application. These concepts are often inherited from the adopted modelling language. For example, the exchange of "signals" between "agents" in SDL [14] may be considered to define an abstract platform that supports reliable asynchronous message exchange. The restricted use of particular constructs in a design language or the use of certain modelling styles or design patterns can serve as a means to select subsets of a language's design concepts.

Instead of implying an abstract platform definition from the adopted set of design concepts for platform-independent modelling, it may be useful or even necessary to define the characteristics of an abstract platform explicitly, resulting in one or more separate and reusable design artefacts. We call this approach *explicit abstract platform definition*. During platform-independent modelling, parts of a pre-defined abstract platform model may be composed with the model of the distributed application. For example, although group communication is not a primitive design concept of UML 2.0, it is possible to specify the behaviour of a group communication sub-system using UML2.0. This sub-system is then re-used in the design of a distributed application. Other examples of pre-defined artefacts that may be included in abstract platforms are the ODP trader [12] and the OMG pervasive services [21] (yet to be defined). The set of design concepts of a design language is still relevant in this approach, since the distributed application and the abstract platform model are described in the language.

In both the implicit and explicit abstract platform definition approaches, there is some overlap between language characteristics and abstract platform characteristics. This leads to the formulation of an important requirement for a design language to support platform-independent design: *the concepts underlying the design language should be precisely defined, so that the characteristics of the abstract platform can be unambiguously derived from these concepts.* This is important for at least two reasons: (1) designers need to know the characteristics of the abstract platform when

defining platform-independent models of an application; and (2) abstract platforms are a starting point for platform-specific realization.

Furthermore, a comprehensive MDA design approach should allow designers to select or define suitable abstract platforms for their platform-independent designs. This leads to the formulation of a second requirement for design languages suitable for MDA: *a design language should enable the definition of appropriate levels of platform-independence.*

In [5], we have discussed how the two approaches to the definition of abstract platforms can be supported using MDA standards, namely UML 2.0 [26] and MOF 2.0 [22].

3.2 Transformation Definition

When multiple levels of platform-independence are adopted, successive (automated) transformations may be used that lead to models at lower levels of platform-independence and, ultimately to platform-specific models (i.e., models at the lowest level of platform-independence with respect to a particular definition of platform).

A transformation is straightforward when the selected target platform (either a concrete or an abstract platform) corresponds (directly) to the source abstract platform. When this is not the case, more effort has to be invested in the transformation.

In general, we distinguish two contrasting extreme approaches to proceed with the design step:

1. *Adjust the target platform*, so that it corresponds directly to the abstract platform.
2. *Adjust the (scope of the) application model during transformation, such that* the requirements specified at source platform-independent level are satisfied by the composition of the application model and target platform model.

In approach 1, the boundary between abstract platform and platform-independent application model is preserved during the transformation step. This implies the introduction of some platform-specific *abstract platform logic* to be composed with the target platform. The nature of this composition depends on the particular requirements for the abstract platform. For example, it may be possible to implement abstract platform logic on top of a concrete platform. Nevertheless, this composition may also imply the introduction of platform-specific (e.g., quality-of-service) mechanisms, possibly defined in terms of internal components of the concrete platform. Extension in a non-intrusive manner is often the preferred way to adjust a concrete platform. Techniques that can be used for non-intrusive extension include interceptors [20], aspect-oriented programming and composition filters [8]. In [6], we have presented an example this approach, using CORBA portable interceptors to extend the CORBA platform with dynamic reconfiguration functionality.

Approach 2 may imply the introduction of (e.g., quality-of-service) mechanisms in the model of the application. This approach may be suitable in case it is impossible to adjust the target platform, e.g., due to the cost implications of these adjustments or the lack of extension mechanisms in a concrete platform.

Figure 3 illustrates these approaches to transformation. We consider a transformation from platform-independent models to platform-specific models.

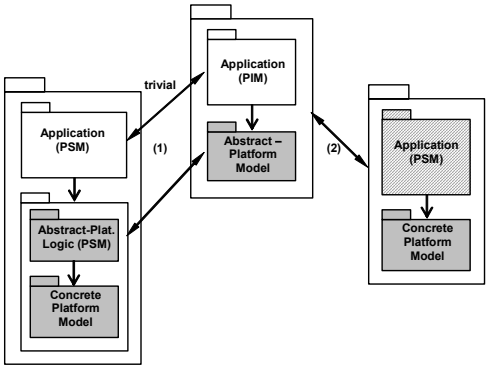

Fig. 3. Alternative approaches to platform-specific realization

Both approaches allow us to target different concrete platforms from the same platform-independent model, with different quality characteristics [2]. Approach 1 can be generalized as a recursive application of service definition (external perspective) and the service's internal design, resulting in a hierarchy of abstract platforms and a concrete target platform. At each step of the recursion, both approaches to transformation can be chosen.

4 Towards a Reference Architecture for Abstract Platform Definition

In MDA development, opportunities for reuse of transformations play an important role in deciding the organization of the execution phase in terms of levels of models and transformations. A single transformation from high-level models to implementations may be costly to develop and is rendered useless in the face of technology platform changes. Given that technology platforms are generally regarded as unstable, it is important to attempt to recognize (intermediate) stable abstract platforms that can be used for a large number of applications. This allows transformations to and from this intermediate abstract platform to be reused.

The proliferation of different abstract platforms reduces the opportunities for large-scale reuse of intermediate models and transformations to and from intermediate models. This calls for the 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 [12]. We believe that 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. An initial discussion on the relation between the RM-ODP concepts and the notion of abstract platform can be found in [4].

An example of the composition of abstract platforms can be found in [5], where we have used UML to combine a number of abstract platforms defined both through the implicit and the explicit abstract platform definition approaches.

5 Related Work

The MDA Guide [21] provides some examples of "generic platform types" and mentions briefly the need for a "generic platform model", which "can amount to a specification of a particular architectural style." Nevertheless, the introduction of these concepts is superficial: for example, the term "generic platform" is not even defined explicitly. In our interpretation of that documentation, we position our notion of abstract platform as subsuming that of generic platform. Abstract platforms can have other relevant characteristics in addition to defining a "particular architectural style", as we have shown in section 3.1. Furthermore, we have focussed on providing guidelines for a designer to define and represent these abstract platforms. The MDA Guide also states that a PIM "exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type." Our concept of abstract platform defines the degrees of platform independence for a PIM.

Explicit abstract platform definition is comparable to the definition of (the behaviour of) connectors in Architecture Description Languages (ADLs), such as Rapide [15], [16] and Wright [1], when considering exclusively the characteristics of interaction support. While the role of middleware platform characteristics in ADLs have been recognized in [18], approaches to systematically separate and relate platform-independent and platform-specific descriptions have not been proposed in the scope of the work on Software Architecture.

6 Concluding Remarks

While, in the context of MDA, much effort has been invested in meta-modelling [22, 23], language definition [26, 28], model transformation specification [24], and tool support, the methodological implications of platform-independence have been largely overlooked. The objective of the Ph.D. work discussed in this paper is to fill this gap by defining a methodology for model-driven design of distributed applications.

We have argued that the architectural concept of abstract platform should have a prominent role in this design methodology. An abstract platform defines platform characteristics that are considered at the particular level of platform-independence, and may also serve as starting point for platform-specific realization.

Design language concepts and characteristics of abstract platforms are interrelated. Therefore, careful selection of a design language is indispensable for the beneficial exploitation of the PIM/PSM separation and the definition of abstract platforms.

Often, some platform characteristics are assumed implicitly in platform-independent designs. This may lead to PIMs that cannot be reused for different platforms or it may lead to PIMs that cannot be directly compared and integrated. It may also lead to transformations that cannot be reused. Platform characteristics assumed in platform-independent designs are better understood and controlled by designers if the characteristics of the abstract platform are explicitly represented in abstract platform definitions.

Work-in-progress includes the elaboration of a reference architecture for abstract platform definition and the application of the proposed methodology and reference architecture in a case study. In this case study, we will define a number of related levels of platform-independence and their abstract platforms, as well as the representation of these abstract platforms in UML. In addition, we will define transformations from the abstract platforms to different concrete platforms.

Acknowledgements

Luís Ferreira Pires, Marten van Sinderen and Chris Vissers should be acknowledged for their invaluable contribution to the Ph.D. work described in this paper. 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 within the MODA-TEL IST project (http://www.modatel.org).

References

1. Allen, R.J., Garlan, D.: A Formal Basis for Architectural Connection. ACM Transactions on Software Engineering and Methodology, Vol. 6, No. 3 (1997) 213–219
2. Almeida, J. P. A., van Sinderen, M., Ferreira Pires, L., Quartel, D.: A systematic approach to platform-independent design based on the service concept. In: Proceedings 7th IEEE Intl. Enterprise Distributed Object Computing Conference (EDOC 2003). IEEE Computer Society, Los Alamitos, CA (2003) 112–123
3. Almeida, J.P.A., Dijkman, R., van Sinderen, M., Ferreira Pires, L.: 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 (2004)
4. Almeida, J.P.A., van Sinderen, M., Ferreira Pires, L.: The role of the RM-ODP Computational Viewpoint Concepts in the MDA approach. In: Proceedings of the 1st European Workshop on Model-Driven Architecture with Emphasis on Industrial Applications (MDA-IA 2004), University of Twente, The Netherlands (2004) 43–51
5. Almeida, J.P.A., Dijkman, R., van Sinderen, M., Ferreira Pires, L.: Platform-independent modelling in MDA: supporting abstract platforms. In: Proceedings Model-Driven Architecture: Foundations and Applications 2004 (MDAFA 2004), Linköping University, Linköping, Sweden (June 2004) 219–233
6. Almeida, J.P.A., van Sinderen, M., Ferreira Pires, L., Wegdam, M.: Platform-independent Dynamic Reconfiguration of Distributed Applications. In: Proceedings IEEE 10th International Workshop on Future Trends in Distributed Computing Systems (FTDCS 2004), Suzhou, China, (May 2004) 286–291

7. Arango, G.: Domain Analysis: from Art Form to Engineering Discipline. ACM SIGSOFT Software Engineering Notes, Vol. 14, No. 3 (1989) 152–159
8. Elrad, T., Filman, R. E., Bader, A. (eds.), Communications of the ACM, Special Section on Aspect-Oriented Programming, Vol. 44, No.10 (2001) 29–97
9. Ferreira Pires, L.: Architectural Notes: a framework for distributed systems development, Ph.D. Thesis. University of Twente, Enschede, the Netherlands (1994)
10. Gavras, A., Belaunde, M., Ferreira Pires, L., Almeida, J.P.A.: Towards an MDA-based development methodology for distributed applications. In: Proceedings of the 1st European Workshop on Model-Driven Architecture with Emphasis on Industrial Applications (MDA-IA 2004), University of Twente, Enschede, The Netherlands (March 2004) 43–51
11. Gray, J., Bapty, T., Neema, S., Schmidt, D.C., Gokhale, A., Natarajan, B.: An Approach for Supporting Aspect-Oriented Domain Modeling. In: Proceedings Generative Programming and Component Engineering (GPCE 2003), Lecture Notes in Computer Science, Vol. 2830, Springer-Verlag (Sept. 2003) 151–168
12. ITU-T / ISO: Open Distributed Processing - Reference Model – All Parts, ITU-T Recommendations X.901, X902, X903, X.904 | ISO/IEC 10746-1, 2, 3, 4 (1995)
13. ITU-T / ISO: Open Distributed Processing - Reference Model - Enterprise Language, ITU-T X.911 | ISO/IEC 15414 (2001)
14. ITU-T: Recommendation Z.100 - CCITT Specification and Description Language. International Telecommunications Union (2002)
15. Luckham, D., Kenney, J., Augustin, L., Vera, J., Bryan, D., Mann, W.: Specification and Analysis of System Architecture Using Rapide. IEEE Transactions on Software Engineering, Vol. 21, No. 4 (1995) 336–355
16. Luckham D., Vera, J.: An Event-Based Architecture Definition Language. IEEE Transactions on Software Engineering Vol. 21, No. 9 (1995) 717–734
17. Microsoft Corporation: Microsoft .NET Remoting: A Technical Overview (2001), available at http://msdn.microsoft.com/library/en-us/dndotnet/html/hawkremoting.asp
18. Di Nitto, E., Rosenblum D.: Exploiting ADLs to Specify Architectural Styles Induced by Middleware Infrastructures. In: Proceedings of the 21st International Conference on Software Engineering (ICSE'99). Los Angeles, CA (1999)
19. Object Management Group: Model driven architecture (MDA), ormsc/01-07-01 (2001)
20. Object Management Group: Common Object Request Broker Architecture: Core Specification, Version 3.0, formal/02-12-06 (2002)
21. Object Management Group: MDA-Guide, Version 1.0.1, omg/03-06-01 (2003)
22. Object Management Group: Meta Object Facility (MOF) 2.0 Core Specification, ptc/03-10-04 (2003)
23. Object Management Group: Meta Object Facility (MOF) Specification, Version 1.4, formal/02-04-03 (2002)
24. Object Management Group: MOF 2.0 Query / Views / Transformations RFP, ad/2002-04-10 (2002)
25. Object Management Group: Unified Modelling Language: Object Constraint Language Version 2.0, Draft Adopted Specification, ptc/03-08-08 (2003)
26. Object Management Group: UML 2.0 Superstructure, ptc/03-08-02 (2003)
27. Object Management Group: UML Profile for Enterprise Distributed Object Computing Specification, ptc/02-02-05 (2002)
28. Object Management Group: Unified Modelling Language (UML) Specification: Infrastructure, Version 2.0, ptc/03-09-15 (2003)
29. Sun Microsystems: Java(TM) Message Service Specification Final Release 1.1 (2002)
30. World Wide Web Consortium: SOAP Version 1.2 Part 1: Messaging Framework, W3C Proposed Recommendation (2003), available at http://www.w3.org/TR/soap12-part1
31. World Wide Web Consortium: Web Services Description Language (WSDL) 1.1, W3C Note (2001), available at http://www.w3.org/TR/wsdl