Innovations Syst Softw Eng (2010) 6:21–28
DOI 10.1007/s11334-009-0117-0

ORIGINAL PAPER

Dealing with variability within a family of domain-specific languages: comparative analysis of different techniques

Ileana Ober · Louis Féraud · Christian Percebois

Received: 30 November 2009 / Accepted: 7 December 2009 / Published online: 5 January 2010

Abstract Almost a decade has passed since the OMG has issued the Model Driven Architecture (MDA) initiative. It soon became obvious that raising the level of abstraction in development and reasoning at the model level would help in asking the right questions at the right time. Based on a concrete problem, we discuss four alternative solutions to a multi-language system design problem. These solutions use a traditional approach, a technique based on modeling, a domain-specific approach, and a mix of modeling and domain-specific techniques, respectively. The solutions depend on the problem, but they are representative for the situations we encounter in practice, therefore giving us a good basis for a larger discussion on the appropriateness of using modeling techniques and on the place of MDA in current software engineering practice and design.

Keywords Model-based development · Domain-specific (modeling) language · Automatic unification · Comparative analysis

1 Introduction

Almost a decade has passed since the OMG has issued the Model Driven Architecture (MDA) initiative [10,26] encouraging the use of models in the development. The MDA/MDD ideas [4,21] are so appealing and so close to the way things go in other engineering areas, that it soon got obvious that they represent the natural way of working, by raising the level of abstraction in development and reasoning at the model level. Modeling is not a 10 years old issue in computer science, as modeling languages (such as SDL [7], statecharts [14] or data flow models [31]) existed and were already used in the 1980s. Nevertheless, MDA has given them a new dimension by advocating their use at a larger scale. As the first concern was to prove the relevance of this approach, the literature abounds with success stories [16], while little was done to criticize [28] or to examine how to avoid over-modeling in order to get the most effective development process. In his guest talk at UML 2003 [3], Bézivin mentioned the lack of modesty through "overselling" MDA as one of the two things that could kill this technique—the other being the lack of ambition.

This paper takes a step in this direction, by showing how the use of modeling is not enough to facilitate a particular system design. We focus on a concrete problem: developing a system made of several components specified using various languages. This case study is issued from joint work with the French National Space Agency (CNES) and can be tackled using several approaches. In this paper we detail a *traditional* approach based on the use of point to point translators that allow to convert components from each language to any other language, a *model based* approach using a common meta-model, and a *problem specific* approach, here based on a categorical approach to unify the domain-specific languages and to generate translators between them. Additionally, we describe a *mixed* approach where we apply modeling techniques in the context of the domain-specific approach.

The solutions are dependent on the problem, but they are representative, therefore giving us a good basis for a larger

I. Ober (✉) · L. Féraud · C. Percebois
IRIT, Université de Toulouse, 118, route de Narbonne, 31062 Toulouse, France
e-mail: Ileana.Ober@irit.fr

L. Féraud
e-mail: Louis.Feraud@irit.fr

C. Percebois
e-mail: Christian.Percebois@irit.fr

discussion on the appropriateness of using modeling techniques.

The rest of this paper is organized as following: Sect. 2 describes the problem that we are trying to solve. Section 3 presents alternative solutions, corresponding to the approaches detailed above. In Sect. 4 we overview a solution mixing the problem specific and model intensive approaches. The analysis of these solutions triggers a discussion in Sect. 5.

2 Case study introduction

The problem we analyse in this paper comes from a case study that we developed with colleagues from the French National Space Agency (CNES) in the context of the DOMINO project [1]. The case study revealed the need to deal with a set of software components specified using related—yet different—domain-specific languages (DSLs).

One major critical issue in this case study is how to handle the differences existing between languages dedicated to the definition of operational procedures for space system testing and operations. Indeed, different space agencies or equipment builders use different languages[1] to remotely control the satellites: Pluto [11] for the European Space Agency, Stol by NASA, and Elisa by EADS/Astrium. These languages are imperative and contain constructs to deal with algorithmic constructs, task scheduling, remote commands, and measures. In spite of the need to occasionally use them jointly, several attempts to impose a unique language have failed, for both economical and political reasons. The various languages of the family differ at the syntactical and semantical level; nevertheless their compatibility is needed in order to have interoperability between agencies and user organizations and it is necessary for the development of a space mission infrastructure.

3 Various solutions

In this section we briefly overview some possible solutions to the problem stated in Sect. 2. We focus on the following categories of solutions anticipated in the introduction: a *traditional* approach to tackle heterogeneity based on the use of point to point translators that allow to convert components from each language to any other language in the system, a *model based* approach using the consensual definition of a unifying meta-model and a *problem specific* approach using a solution that is very dependent on the problem to be solved.

3.1 Traditional approach

A straightforward approach consists in the development for each pair of languages of a compiler in charge with the

[1] The specifications for most of these languages are not public.

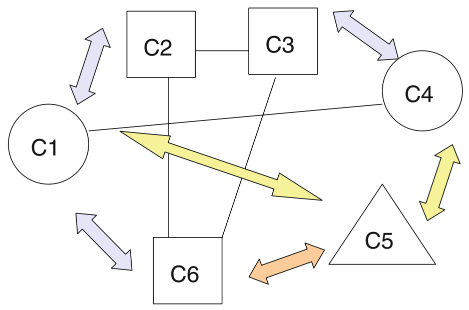

Fig. 1 Overview of the traditional technique

translation from the source language to the target one. The toil required to craft such compilers is generally dependent on the richness and the distance between both of the languages: the poorer and the closer they are, the easier the compiler construction is. In fact, DSLs designed to address the same domain of interest generally offer the two above conditions: DSLs syntax and semantics are weaker than the ones of a general purpose oriented language (limited richness) and, although different, their features are similar (short distance).

One optimisation would be to use an intermediate *pivot* language. This solution is very close to the one we will discuss in Sect. 2 at model level, therefore we will not investigate here any further.

In Fig. 1 we illustrate the principles of this traditional approach. The system under development is made of six components C1 to C6 specified in different languages, corresponding to their shape. The components described in a same language (identically shaped), can be combined directly; this is shown as simple lines between the corresponding components. The composition is done using a composition mechanism that is outside the scope of this paper. In Fig. 1, the translators are represented as full arrows. We use the same hatching for arrows corresponding to a same translator. Therefore in our case we need three bidirectional translations. In [6,5], while addressing the same problem of unifying languages for spacecraft operations, the authors propose a method and a tool (APPAREIL) to automate the process of generation translator from a specification of their language grammar annotated by semantics related information. The core of the APPAREIL system relies on a translator builder aiming at generating a full source to target translator. When building this translator, APPAREIL rests on the syntax of both of the languages and some tagged annotations. In order to verify the correctness and completion of the structure, the ASF+SDF meta-environment [19] is used. The annotations are used to specify the correspondence between languages features in the two languages. In order to prevent the cases where some mismatches between languages occur and an automatic translation based on annotations fails, APPAREIL provides relevant informations extracted from an analysis of the notations. This information is then processed by a dedicated library which composes these additional transformations and include them in the translator. The approach described above to solve DSLs interoperability is significant in the sense where the relation between DSLs is viewed in terms of grammar engineering.

3.1.1 Advantages

Addressing the problem in terms of *grammar engineering* allows to take advantage of a lot of previous works. Among them, it is possible to quote the addition of new features to language in terms of attribute grammars [29], the production of translators resting on syntax directed rules sets [30], linglets based translation [9]. More generally this approach profits by half century of research in compiling [2].

3.1.2 Drawbacks

As pointed out previously a lot of results and tools are available to craft one to one translators with reasonable effort, taking into consideration simplicity of DSLs syntax and semantics. Nevertheless, a specific translator has to be built and maintained for every language pair, which constitutes the main disadvantage of this approach.

3.2 Model-based approach

This solution we present in this section from the observation that the major issue in the case of our problem is the need to conduct the development using a set of languages that cover the *same* domain. Therefore, one can define a domain *meta-model*. The various languages would then be only *instances* of this meta-model.

This solution corresponds precisely to an OMG initiative for defining a meta-model for spacecraft operations [24]. The objectives of such a metamodel are detailed in [8] and range from making less expensive the transfer of a satellite from one ground system to another (each of these using their own proprietary languages, which often differ), to increasing the interoperability between ground systems. The rational of having such a meta-model are detailed in [8].

This meta-model would give an instrument to describe an entire domain, practically leading to a kind of *unifying language for spacecraft operations* descriptions. In Fig. 2 we show an example of a system composed of six components described in various languages. As before, C1 to C6 are the different components that need to interact in the context of a unique system. Different shapes for components correspond to different languages used for the development of these components. In our case, the interoperability is achieved by means of translators from the base languages of these components to a *unified* one. The star-like shape represents

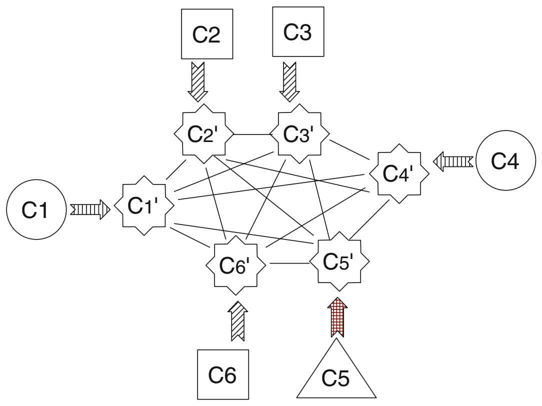

Fig. 2 Heterogenous components interoperating via a unified language

the unifying language, towards which we need translators (at least unidirectional, but most likely bi-directional) from the original languages. These translators are represented by arrows.

One can imagine two kinds of usage for such a consensual unified language: define bi-directional translators from existing languages to the unifying one, and replace existing languages with the unifying one (this would correspond in Fig. 2 to a situation where we would directly have the components C′).

One can notice the similarity that exists between this situation and the definition of the Unified Modeling Language at its time, that emerged as a unification of a set of more or less related languages.

3.2.1 Advantages

The existence of such a meta-model would increase the comprehension of the field, by making explicit relationships that exist between the various concepts. Therefore such a meta-model would be beneficial to the various equipment builders, regardless of the base language they are supposed to define. It would allow them to have a full picture of the concepts both of them used directly and of those that may be not necessary to them.

In case the unifying language is used as a middleware between the concrete languages, its existence reduces the number of needed translators to $O(n)$, while in the case the unifying language is adopted it can be the only language used therefore eliminating any issue related to interoperability for new code development. Note however that in the case of legacy code, even if the adoption of the unified language was complete, we still need the translators.

3.2.2 Drawbacks

As it has to cover an entire domain, the unifying meta-model/ language is necessarily more complex than any individual

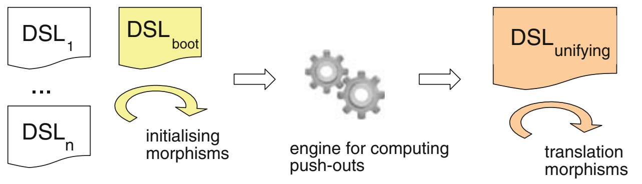

Fig. 3 Obtain the unifying language in a categorical approach

one. This is the reason why in Fig. 2 we use the most complex shape for components described in this language. Applying validation and verification in this context is therefore more difficult.

The raise of a new language would induce additional costs related to training the teams on this new language, to the deployment of this new language and to the maintenance of the legacy code.

Although the OMG RFP was released early 2005 with answers expected late 2005, at the time of writing this paper (3 and a half years later) there is no official submission recorded by the OMG, in spite of the efforts deployed to define a unified meta-model and of the existence of a real market for it. Actually, economical and political reasons render the definition of a such a meta-model/language very difficult.

A variant of this model based approach is a solution that consists in using rather than a meta-model, an intermediate pivot language. The consensual definition of such a language is as difficult as the definition of a meta-model, therefore this option is in practice equivalent to using a domain meta-model.

3.3 Problem-specific approach

In this section, we propose a solution very dependent on the concrete problem we are about to solve. In most of the cases, such an approach exists, although its details will vary. This solution, presented into more detail in [22], starts from the formal specification of each of the domain-specific languages that need to be used in our context.

The principles of this approach are overviewed in Figs. 3 and 4.

Using classical results in the category theory we can *construct* the formal specification of a unifying language. This approach is based on results in category theory [12] that ensure that the colimit in this context exists and that it has the desired properties. By computing colimits on the category of algebraic specifications that define the semantics of each DSL in the family, we construct a language that unifies the family.

As illustrated in Fig. 3, the formal specification of each DSL, that covers both the syntax and the semantics, is considered as an object in the category of algebraic specifications. In order to have all the premises needed for computing the

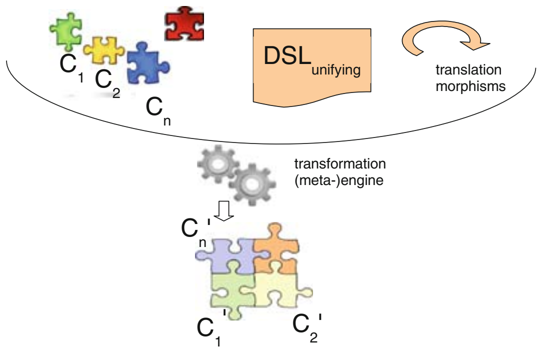

Fig. 4 Components interoperability via the obtained unifying language

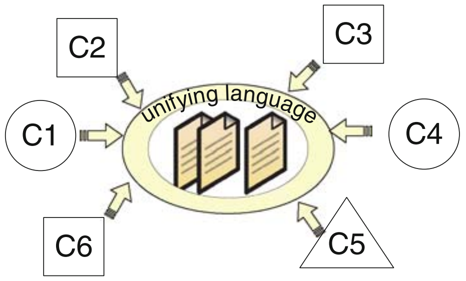

Fig. 5 Overview of the problem-specific approach

colimit, we need to additionally define a *boot object* and a set of *initializing* morphisms from this object to each of the individual DSL objects, that ensures the existence of a common source. The boot object can be seen as a repository that contains the concepts common to the various objects.

In the case of objects corresponding to algebraic specifications of a family of languages, the boot object gathers the concepts present across all the languages in the family to unify. For more details on the role and content of the boot object and of the initializing morphisms the reader is referred to [22].

In Fig. 5 we illustrate the principles of this approach. In order to achieve interoperability, we use a unifying language whose definition is obtained *automatically*, together with translators from each source language.

3.3.1 Advantages

Classical results in category theory [12] allow obtaining *by construction* the formal definition of a unification language in terms of algebraic specifications, as well as translators from the source DSLs to this unification language. This language has the advantage of being automatically built as the algebraic specification forms a finitely co-complete category [12].

The fact that we also obtain translators from the original languages to this one, allows its potential transparent use.

This is a key issue in a context where one of the main benefits of using DSLs is that they allow a better exploitation of domain experts experience. It is essential that the end users continue to use the environments and tools specific to each language, while being able to switch easily between them.

This approach was tested in a setting made of already existing tools. In particular, we used Specware [18] that gives us a framework to work with categories and ASF+SDF [19] to use the morphisms and the unifying language specification in order to translate code from a specification in a DSL to a specification in the unification language.

Last but not least, properties established in the context of DSLs and expressed as theorems can be transferred to the unification language. Thanks to the categorical approach, a property defined in one DSL is transferred to the unifying language through morphisms: a specification morphism translates theorems of a domain specification to theorems of the codomain. The main benefit is to give the opportunity to verify properties separately in the context different DSLs and then to gather them all in the unification DSL framework.

3.3.2 Drawbacks

The language unification uses category theory as underlying mechanism. In order to ensure that the unification object goes beyond a mechanical union of the language concepts present in the various languages, we have to express somewhere the correspondence between related constructs that exist in various languages in order to minimize the perpetuation of related concepts. The boot object contains the formal specification of the constructs that exist across the various languages. The mapping between the concepts in the boot object and their correspondents in the specific DSLs is captured by the initialization morphisms that define the each DSL in terms of boot extensions. The definition of the boot object and of the initialization morphisms is the price to pay in order to obtain the automation. In previous work we have given guidelines that facilitate these definitions [22].

This approach covers the algebraic specification. It is unrealistic to expect that a end user would directly interact with this. Therefore additional techniques are needed in order to make the use of algebraic specifications transparent.

4 Integrating modeling and domain-specific approaches

The solutions described in Sects. 3.2 and 3.3 contain promising ideas, with complementary drawbacks. In this section we describe a solution that aims at combining their advantages.

The most important gain of the domain-specific approach (Sect. 3.3) is that domain engineers can work in their usual development environment and therefore they can concentrate on the problem to solve and exploit their own expertise. In the case of the model based solution (Sect. 3.2), one of the main benefits of having a domain meta-model is that it would ease the global reasoning and comprehension. Such a meta-model can be useful if a global model of an application is suited, combining the parts specific to spacecraft operations with other parts of a system.

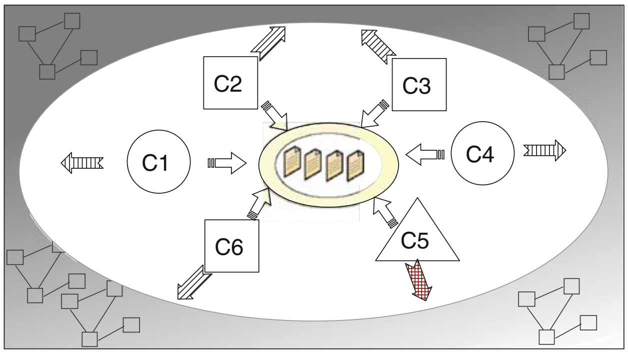

Fig. 6 Problem-specific approach integrated into a global model

The combination of these two approaches can result in using model-based techniques for coarse-grained aspects and domain-specific techniques for the fine-grained ones. For this, we need both a domain meta-model as the one that is being defined by the OMG [24] and domain-specific techniques, as the ones described in Sect. 3.3.

Figure 6 illustrates the principles of this solution. We start from the observation that one of the main assets in software development is the expertise of the people developing the low level parts of the application, which in our case are engineers with minimal training in programming. An acceptable solution should build on this hypothesis.

The mixed solution consists in using the problem specific approach for achieving interoperability, in parallel with the use of a system model, whose role is not to ensure interoperability, but to get the global view of the system. The existence of translators (or model transformers) towards this unified language/meta-model from a domain-specific one, would ensure the accuracy of the global models with respect to the application level reality.

One can notice that we pass information from the application level to the model level. The reason for this is that we believe that model-intensive techniques should be a complement to the traditional ones, rather than their substitute (at least not from the beginning). Each of them has its specific advantages, we find it natural to try to combine the two. In the long run, it may be opportune to extend the use of model intensive techniques to the application level. However, we believe the domain experience should remain an important part of the picture.

4.1 Advantages

The use of the mixed approach allows domain engineers to keep their working environments. The presence of a unified meta-model/language allows to get a more general picture of the domain, and to include the part of the application specific to spacecraft operations into a more general application model. In this approach the use of the legacy code is not problematic, as no change in the low level development is implied.

The approach allows a progressive migration to model intensive approaches, which in our opinion is more honest and more realistic than a more revolutionary one. The two approaches are compatible and as a result it is possible to translate DSLs to models, thus ensuring the existence of a bridge between the two *paradigms* [13].

4.2 Drawbacks

The most important drawback is that the partial introduction of models, by not allowing to exploit them completely, may get people reluctant to their added value and could slower the deployment of MDD.

By re-using the problem specific approach for the application level, we get on the drawbacks of using this problem-specific approach. In our case, the most important is the fact that we have to additionally define the initial language and the initializing morphisms. However, the existence of a domain meta-model facilitates the task of finding commonalities between subsets of the same domain.

One important challenge when integrating domain-specific techniques in a system level modeling consists in choosing the right domain level information that has to be abstracted at the system level. This would allow different domain engineers to effectively communicate and exchange information.

5 Discussion

In this section, we will examine some factors that may influence the choice of the most appropriate technique to be used for dealing with families of domain-specific languages. The factors we will consider here are: *the project stage*, the *development team composition* and the *degree of variability* that needs to be accommodated.

With respect to the *project stage* we can distinguish between new projects and ongoing development projects or projects where the part of existing code is largely preponderant. One can notice that the number of projects that are based only on new code is ever decreasing. As stressed in [20]

"*[...] much software actually is potentially common and generic [...] Results show that software redundancy is as high as 85% [and a cited study] found that 75% of application code, 50% of system program, and 70% of telecommunication program shared identical code or functions*"

As mentioned above, the genuinely *new projects* are rare. In their case, the use of modeling techniques can fit much more smoothly into the overall development, because there is no issue of backwards compatibility.

We expect the effort required for the introduction of MDA techniques in new projects would mainly depend on contextual factors such as the team and project manager training and experience.

Another influencing factor is the nature of the domain to be applied to. The more the domain is close to a specific engineering field, the more the weight of domain expertise is important, thus applying modeling, may be difficult.

Most of the energy is devoted in the context of *projects with important reused parts*. In our example, although new developments are constantly being made, the proportion of using existing code, or existing application is very important. In the case of such projects, based on the solutions exposed in the previous sections, we claim that a step-by-step introduction of the MDA techniques would be the best strategy. Indeed, this strategy would allow a gradual experience gain and team training, which are needed before a more complete deployment of MDA.

If we consider the *development team composition*, the more software engineers are present in the team, the more a solution intensively exploiting MDA, gets interesting, such the ones presented in Sects. 3.2 and 4. On the other side, in the presence of an important team of domain engineers, the capitalisation of their expertise is essential. Thus the solutions that would allow them to work in their typical domain-specific environment would be preferable. This corresponds to the solutions presented in Sects. 3.1, 3.3 and 4.

As one can notice, in case we have both important teams of software engineers and domain engineers, the mixed solution described in Sect. 4, seems to be the most reasonable choice.

Within a project that needs to deal with a specifications done using domain-specific languages on a same family, the set of the base domain-specific languages may be fix or it may vary. The pairs of languages that are to be used in conjunction may be known in advance or may vary over time. We can also have a complete flexibility where the set of languages is subject to dynamic change.

If the set of domain-specific languages is fixed and we know exactly between which pairs of languages we need translators, the solution described in Sect. 3.1 may be a sufficient choice. One can notice however, that it seems unrealistic to assume that we know beforehand all possible usages and that no new language will be needed.

If new languages may be added dynamically or if we do not know which pairs of languages are to be used in conjunction, the most flexible solution is the one described in Sect. 3.3, generalised by the one described in Sect. 4, as according to these solutions the unifying languages is dynamically computed based on the set of the language actually used.

With respect to *variability*, it worth mentioning one of the most recent OMG initiatives [25] that targets the definition of an (abstract) language to deal with the variability between related domain-specific languages and within product lines. At the time we write this paper no official proposal exists for this, however in the literature we can find some work in this direction. In [15] and [23] the authors propose an approach to specify the variability in product lines based on the use of feature diagrams [17] that leads to a complete model of the expected functionalities. Moreover, in [27], the authors use this approach to automate the code generation for a train control language.

Given the history of OMG standards (in particular with respect to UML, the action semantics proposal and with MARTE) it is very unlikely that this standard would go beyond a (more or less precise) description of the abstract syntax of the *common variability language*.

We believe, that in order to be able to address effectively the variation between families of languages, and therefore allow for automatic translation and incremental verification, we need more than just a language (abstract) definition. A formal technique, such as the one we present in Sect. 3.3, integrated in a system level development approach such as the one presented in Sect. 4, would allow for effective tool support.

6 Conclusion and future work

In this paper we consider four alternative solutions to the interoperability issue in the case of a system built using components developed using various (related yet different) domain-specific languages. This example is extracted from a case study that we worked on jointly with colleagues from CNES (The French National Space Agency). The considered solutions correspond to: a *traditional* approach—based on the use of point to point translators that allow to convert components from each language to any other language in the system, a *model based* approach—using the consensual definition of a unifying meta-model; a *problem-specific* approach—in this case using a categorical approach to unify the original domain-specific languages and to generate translators between them, and finally a *mixed* approach—where we combine modeling techniques with a domain-specific approach.

The analysis of these various solutions offers a starting point for a discussion on the appropriateness of using modeling techniques and on the place of MDD in current software engineering practice and design.

One of the most recent OMG initiatives [25] targets the definition of an (abstract) language to deal with the variability between related domain-specific languages. We believe, that in order to be able to address effectively the variation between families of languages, and therefore allow for automatic translation and incremental verification, we need more than just a language (abstract) definition. A formal technique, such as the one we present in Sect. 4, integrated in a system level development approach such as the one presented in Sect. 3.3, would allow for effective tool support.

References

1. Domaines et processus méthodologique (2009) http://www.domino-rntl.org/
2. Aho AV, Sethi R, Ullman JD (1986) Compilers: principles, techniques, and tools. Addison-Wesley, Reading
3. Bézivin J (2003) MDA: from hype to hope, and reality. In:Invited talk at 6th international conference on UML, San Francisco, CA, USA, October 20–24
4. Bézivin J, Gerbé O (2001) Towards a precise definition of the OMG/MDA framework. In: ASE. IEEE Computer Society, pp 273–280
5. Camacho DO, Mens K (2008) Appareil: a tool for building automated program translators using annotated grammars. In: ASE. IEEE, pp 489–490
6. Camacho DO, Mens K (2007) Using annotated grammars for the automated generation of programs transformers. In: Ingénierie Dirigée par les Modeles
7. CCITT (1988) Specification and description language. Recommendation Z.100, Blue Book
8. Chaudhri G, Cater J, Kizzort B (2006) A model for a spacecraft operations language. In: SpaceOps 2006. American Institute of Aeronautics and Astronautics. AIAA 2006–5708
9. Cleenewerck T, D'Hondt T (2005) Disentangling the implementation of local-to-global transformations in a rewrite rule transformation system. In: Haddad H, Liebrock LM, Omicini A, Wainwright RL (eds) Symposium on applied computing. ACM, New York, pp 1398–1403
10. DSouza D (2001) OMG's MDA, An Architecture for Modeling. Technical report, OMG
11. European Space Agency (2008) ESA. ECSS-E-ST-70-32C—Test and operations procedure language Standard Document
12. Goguen JA, Burstall RM (1992) Institutions: abstract model theory for specification and programming. J ACM 39(1):95–146
13. Göktürk E, Naci Akkøk M (2004) Paradigm and software engineering. In: Turkey Ege University, Izmir. Impact of Software Process on Quality (IMPROQ) Workshop, May 2004
14. Harel D (1987) Statecharts: a visual formulation for complex systems. Sci Comput Program 8(3):231–274
15. Haugen Ø, Møller-Pedersen B, Oldevik J, Olsen GK, Svendsen A (2008) Adding standardized variability to domain specific languages. In: SPLC. IEEE Computer Society, pp 139–148
16. Hössler J, Born M, Saito S (2006) Significant productivity enhancement through model driven techniques: a success story. In: EDOC. IEEE Computer Society, pp 367–373

17. Kang K, Cohen S, Hess J, Nowak W, Peterson S (1990) Feature-Oriented Domain Analysis (FODA) Feasibility Study
18. McDonald J, Anton J (2001) SPECWARE—Producing Software Correct by Construction. Kestrel Institute Technical Report KES.U.01.3
19. Klint P (1993) A meta-environment for generating programming environments. ACM Trans Softw Eng Methodol 2(2):176–201
20. Lim WC (1998) Managing software reuse: a comprehensive guide to strategically reengineering the organization for reusable components. Prentice-Hall, Inc., Upper Saddle River
21. Miller J, Mukerji J (2003) MDA guide version 1.0.1. omg/2003-06-01. Technical report, OMG
22. Ober I, Dib AA, Féraud L, Percebois C (2008) Towards interoperability in component based development with a family of DSLs. In: Morrison R, Balasubramaniam D, Falkner KE (eds) ECSA. LNCS, vol 5292. Springer, Berlin, pp 148–163
23. Oldevik J, Haugen Ø, Møller-Pedersen B (2009) Confluence in domain-independent product line transformations. In: Chechik M, Wirsing M (eds) FASE. Lecture notes in computer science, vol 5503. Springer, Berlin, pp 34–48
24. OMG (2005) RFP Spacecraft Operations Language Metamodel. Technical report, OMG
25. OMG (2009) Draft RFP Common Variability Language. Technical report, OMG. ad/2009-08-07
26. Soley R (2001) Model-Driven Architecture. Technical report, OMG
27. Svendsen A, Olsen GK, Endresen J, Moen T, Carlson E, Alme K-J, Haugen Ø (2008) The future of train signaling. In: Czarnecki K, Ober I, Bruel J-M, Uhl A, Völter M (eds) MoDELS. Lecture notes in computer science, vol 5301. Springer, Berlin, pp 128–142
28. Uhl A, Ambler SW (2003) Point/counterpoint: model driven architecture is ready for prime time/agile model driven development is good enough. IEEE Softw 20(5):70–73
29. Van Wyk E, de Moor O, Backhouse K, Kwiatkowski P (2002) Forwarding in attribute grammars for modular language design. In: Nigel Horspool R (ed) Compiler construction. Lecture notes in computer science, vol 2304. Springer, Berlin, pp 128–142
30. Wile DS (1991) Producer of parsers and related tools system builders' manual 1994 edition draft. Technical report, USC Information Sciences Institute
31. Yourdon E, Constantine LL (1979) Structured design: fundamentals of a discipline of computer program and systems design. Prentice-Hall, Inc., Upper Saddle River