On the integration of domain-specific and scientific bodies of knowledge in Model Driven Engineering

Eric Miotto, Tullio Vardanega
University of Padua
Dept. of Pure and Applied Mathematics
via Trieste 63, 35121 Padua, Italy
ermiotto@studenti.math.unipd.it, tullio.vardanega@math.unipd.it

***Abstract*—With the adoption of Domain Specific Languages (DSL), Model Driven Engineering (MDE) recognizes that application domains influence the way we specify and construct software systems. Notwithstanding their domain-specific nature, application systems often share common transversal concerns – for example, automotive, railway and aerospace domains alike require checking that critical temporal deadlines are met. It is not very wise therefore that every application domain should resolve transversal issues on its own: wherever possible instead, recourse should be made to the common body of knowledge that addresses the problem and solves it in a manner sanctioned by the scientific community.**

An effective MDE development environment should therefore subsume two distinct bodies of knowledge: the domain-specific one and its scientific complement. We are interested in devising a provably correct and affordable way to implement such an MDE environment. Two important questions however stand before us in that endeavor:

1. **What language should we employ to enable the modeling of both bodies of knowledge: should we go for either UML profiles or metamodeling techniques?**
2. **What factors most impact the complexity and cost of proving model transformations correct?**

In this short paper we limit ourselves to elaborating on these problems, underpinning them with some background reasoning on their fundamental role to the very heart of MDE.

***Index Terms*—model-driven engineering, metamodel, model, transformation, composability, compositionality, UML profile, Platform Specific Model, Platform Independent Model, application domain, scientific body of knowledge, semantics, correctness, Unified Modeling Language 2, Meta Object Facility, Domain Specific Language, language**

I. Introduction

In the quest for increased quality and productivity, Model Driven Engineering (MDE) promotes [1], [2]:

1. the use of models at various levels of abstraction as a vehicle for system specification, in the place of source code artifacts and informal diagrams that do not qualify as models;
2. the use of automated transformations to progressively turn the user model into a software product ready for final compilation, binding and deployment.

In this paper we borrow concepts and terminology from a particular MDE initiative, the Model Driven Architecture (MDA) standard [3], [4] endorsed by OMG, that for instance was adopted in the ASSERT project (http://www.assert-project.net/). MDA uses models to abstract away from technological choices and to facilitate the port to the specific target. To this end, MDA advocates the following approach (figure 1):

1. we first construct a Platform Independent Model (PIM) that specifies the solution in a way that does not depend on any particular implementation;
2. we then transform this model into a Platform Specific Model (PSM), by feeding the transformation with information about the chosen execution platform and its characterizing parameters; we assume that the PSM is automatically generated and users are not allowed to directly modify it;
3. we perform a range of analyses on the PSM to validate its feasibility and adequacy against a range of criteria and, when satisfied, we launch the final stage of transformation from PSM to code, otherwise we return to the PIM.

MDE acknowledges the centrality of application domains to software development. As a corollary of this observation, models might well be expressed using Domain Specific Languages (DSL): those languages in fact may be specifically tailored to the expressive needs of the application domain of interest.

In this particular respect, DSL seem to have some advantages over General Purpose Languages (GPL) like UML and, even more so, mainstream programming languages:

- a DSL contains concepts apt for the target domain and domain specific tools can better understand the intended semantics, thereby permitting to detect behaviour that does not fit the domain. Conversely, GPL contain general concepts, usually related to computation. We can obviously express domain specific concepts with them, but the semantics of the domain – the one we are ultimately interested in – is buried under the semantics of computation. Hence program comprehension and verification become more problematic.
- DSL can be used directly by domain experts, while GPL can be used only by software developers and the two job descriptions may vastly differ.

Application domain knowledge alone however is not sufficient. Let us consider for example the construction of software for automotive, railway and aerospace domains. We are clearly considering three different application domains. Yet, these application domains face common concerns, (for instance) in

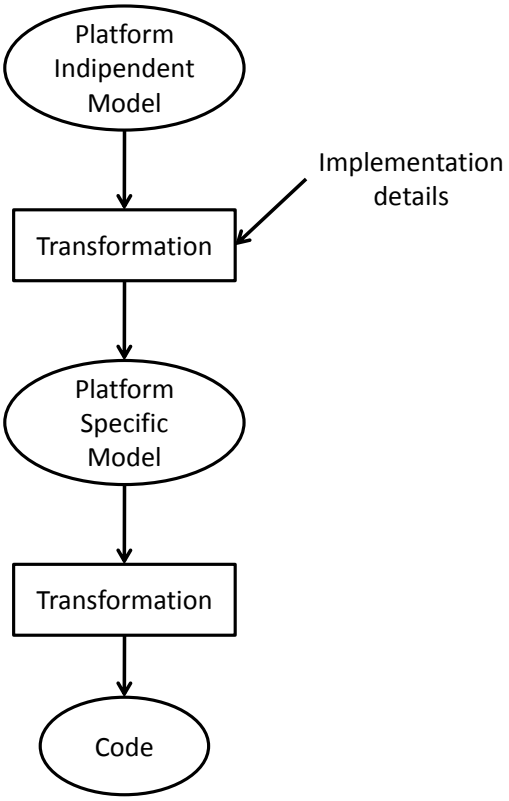

Figure 1. In MDA the system is specified using a Platform Indipendent Model (PIM), which abstracts away from implementation details. In order to obtain a system that can be turned into a deployable executable, we have to transform the PIM into a Platform Specific Model (PSM). This transformation requires guidance information on what implementation choices to make. A final stage of transformation is required to turn the PSM into source code.

that parts of their software must demonstrably meet temporal deadlines. It does not seem especially wise that each application domain should develop their own custom solutions to common transversal problems, known to science and equipped with standard solutions:

- effort is surely wasted in insisting to confront over and over again problems that have been solved before;
- recognizing common transversal problems requires knowledge that is more scientific than domain specific; the former should be leveraged if the solution needs to be sound, solid and cost effective.

When we for example consider timeliness and predictability problems, we ought to know that the real-time scientific community has long known those problems and devised a range of analytic and engineering techniques to best cope with them. Such techniques may be regarded as best practices that form a so called *scientific body of knowledge*. Our two main contentions in this paper are that: (i) the domain specific knowledge should be complemented with adequate doses of scientific body of knowledge; and (ii) the MDE paradigm is best suited for leveraging that knowledge (figure 2).

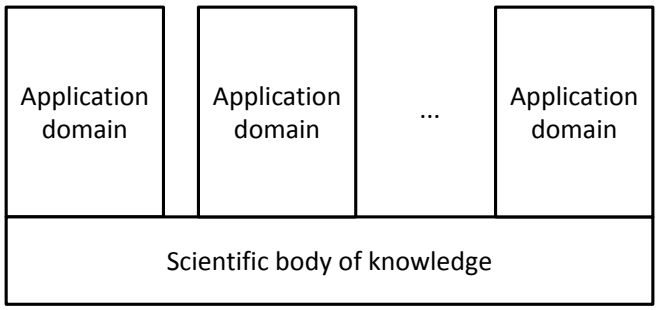

Figure 2. A scientific body of knowledge contains common concepts that are employed by several application domains.

To summarize, both application domain knowledge and scientific bodies of knowledge are crucial to effective MDE and they should both guide and steer the application development.

In the MDA approach in particular, those two complements of knowledge have the role and impact illustrated in figure 3:

- they determine how the DSL to use for PIM and PSM are defined. At PIM level we must be able to conveniently express concepts that pertain to the application domain. Conversely, at PSM level we must be capable of constructing the system using methods that have a scientific pedigree and background;
- the application domain often requires to deploy applications in accord with some estabilished reference architecture, which may be physical as well as logical;
- the transformation of the PIM to PSM is guided by both application domain and scientific knowledge. The former retains design choices favored by the user (or the domain culture), while the latter permits to implement them in a scientifically based manner.
- the allowable range of analyses to be performed on the PSM and the practices of code generation from it must be rooted in the scientific body of knowledge.

We want to be able to build an MDE environment and infrastructure that equally leverages on application domain knowledge and scientific knowledge in supporting effective and economic construction of correct-by-construction software systems.

Two problems stand before us in the attainment of this goal; in the remainder of this paper we want to elaborate on their essence, background and trade offs:

1. DSL seem to be important vehicles for the expression of both the application domain and the scientific bodies of knowledge. Given the coverage that both elements have in the deployment of MDA, DSL should serve for both PIM and PSM. The DSL should be carefully designed, for they determine the effectiveness at the user level and the affordability of the implementation and the durability of the infrastructure. DSL can be defined through UML profiles or through a metamodel specified with some metamodeling language (for example MOF). Which way to go here is not clear, though. In section II we develop some reasoning

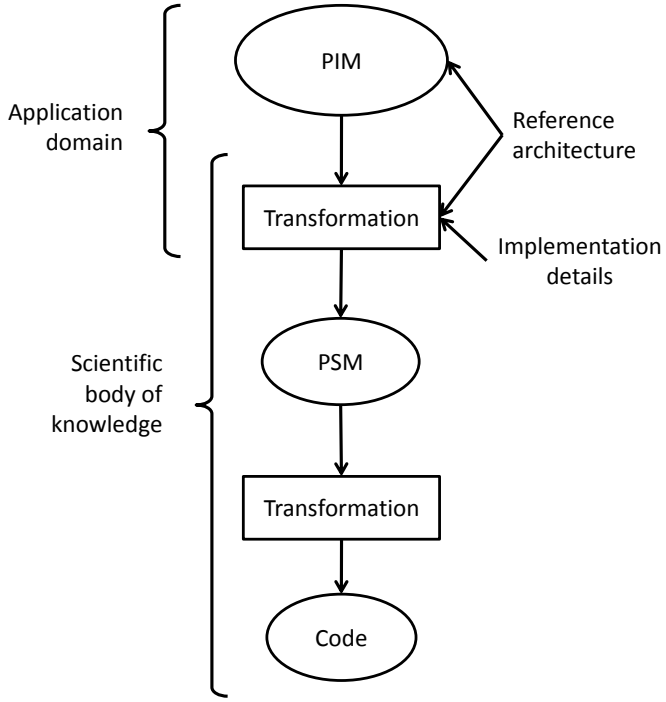

Figure 3. The way application domain knowledge and scientific knowledge fit in MDA. PIM and its languages are influenced by the application domain; PSM and its languages are influenced by the scientific body of knowledge; the application domain may impose a reference architecture; the transformation from PIM to PSM is determined by both application domain and scientific body of knowledge; analyses of and code generation from the PSM are influenced by the scientific body of knowledge.

about this particular problem;

2. as we have already noted, the application domain and the scientific body of knowledge influence the entire spectrum of model transformations (from PIM to PSM and from PSM to code). It is obviously imperative that the transformations are proved correct for some definition of correctness. As the proof of correctness may be considerably onerous we are interested in taming its complexity to the maximum possible extent. We must therefore understand what factors most influence the relevant costs: in section III we single out three such factors, which concern (i) the number of views defined on the PIM, (ii) the incremental construction of the PIM, and (iii) the number of metamodels used to support views and models in both PIM and PSM.

II. Problem One: Which DSL

DSL form an important foundation for MDE, because they permit to express concepts rooted in application domains and in scientific bodies of knowledge. Deciding how DSL are defined is a difficult design decision, which effects the very construction of MDE infrastructure. There are two main ways to define a DSL:

1 to specialize the UML 2 metamodel through the definition of a UML profile;

2 to create a new metamodel from scratch using MOF or other metamodeling languages.

A thorough comparison between UML profiles and metamodeling can be found in [5], in which metamodeling is rated more powerful but also less supported by tools than profiles. We think that the answer is not so unequivocal and that there are other considerations that are worth making:

- how UML profiles are defined in UML 2;
- whether the way MOF and UML are defined suggest which is best for defining DSL;
- in which way tools allow to create DSL.

Let us discuss these issues in isolation.

A. Definition of UML profiles

The way profiles are defined in UML 2 seems a little fuzzy to us. In our opinion, this fuzziness stems from the following factors:

- profiles are described in both Infrastructure [6] and Superstructure [7]. The respective descriptions are almost the same (except for the graphical notation added in Superstructure) and we see no value in the duplication;
- the description of profiles is poor. To us the specification proved very difficult to parse, in that it is written in a confused way and it is not well structured;
- at first glance, profile have mechanisms to target only a subset of the UML metamodel – in particular the visibility rules enforced by the metamodelReference and the metaclassReference associations of the Profile metaclass. In fact, it seems to us that they don't accomplish that goal:
  - metamodelReference is limited to the compliance levels and the packages defined in UML – we can still import unwanted metaclasses;
  - metaclassReference seems to address the latter concern, but its use can be quite laborious, especially if we want to leave out a few metaclasses;
  - last but not the least, the entire visibility mechanism seems a futile exercise, since in the specification after its description we find the following clause: "The filtering rules defined at the profile level are, in essence, merely a suggestion to modeling tools on what to do when a profile is applied to a model." In our opinion, this means that all the mechanism is optional (for instance, we didn't find any such mechanism in Papyrus UML [8]) and in fact in the profile definition we are referencing the entire UML metamodel, which may well be an unwelcome burden.

In summary, we contend that the current definition of UML profiles has flaws that hamper their comprehension and their use.

B. MOF and UML

It is certainly worth investigating and clarifying the relationship between MOF and UML, in order to understand which is best at creating DSL.

This might at first seem an easy question. Indeed, there has been a time when UML and its profile mechanism were

promoted as the sole standard way to define DSL. When the MDA initiative proposed MOF as the standard base to specify the metamodels for modeling software systems, however, the opinions changed as some had reasons to prefer MOF to UML for specifying DSL.

In fact the situation isn't clear either way, unfortunately. To begin with, MOF and UML were actually born together: both the Request For Proposals (RFP) for MOF [9] and the one for UML [10] were issued simultaneously in 1996, though in different contexts – MOF for CORBA, UML for modeling. OMG soon acknowledged that UML could be metamodeled with MOF and it was wise to use the same core constructs for both languages. Indeed the RFP for UML 1 required to furnish a mapping between MOF and UML constructs and the UML 1.1 proposal [11] has it.

The specifications of both MOF 2 [12] and UML 2 Infrastructure [6] bring this integration to maturity. As required by their RFP ([13] and [14] respectively), considerable effort has been devoted to develop a core suitable for both languages, which contains (i) basic data types, (ii) abstract concepts needed in the definition of metamodels and (iii) concrete constructs related to object oriented modeling. This approach has at least two advantages according to [12]:

- it avoids the creation of another modeling language. UML is widely known and moreover its base concepts are apt to model modeling languages;
- it eases the creation of metamodeling tools through the adaption of existing UML tools. In a broader sense, UML tools are already capable of metamodeling because they already got the necessary language core.

Put otherwise, MOF is a formal way to say that UML is capable of metamodeling. MOF and UML are thus equally important: UML provides the modeling notation and MOF adds facilities useful for metamodel management. In some sense, it can be seen as an incarnation of the UML 2 proposal endorsed by the Distributed Systems Technology Center [15].

C. Tools for creating DSL

So far we have only considered methodological arguments, but it is useful to also have a look at tools for creating DSL. In theory the methodology should drive the technology: but it often happens that tools favor some methodology and make more expensive others. Thus we should look at what way the existing tools let us create DSL, to gage which way they lean, whether for UML profiles or for metamodels.

We first searched for tools suited for creating DSL in the research literature, on Model-Driven Development Tool Implementers Forum home page (http://www.dsmforum.org/events/MDD-TIF07/) and on Johan den Haan's microblog on MDE (http://twitter.com/ModelDriven). The tools we found include: Eclipse EMF [16]; MetaCase MetaEdit+ [17]; Vanderbilt University Generic Modeling Environment [18]; MOFLON [19], JetBrains Meta Programming Systems [20]; Microsoft DSL Tools [21] and Microsoft "Oslo" [22]. All these tools use metamodeling in order to specify new languages.

On the other hand, OMG has defined several DSL using profiles and these DSL are supported by industry. For example, MARTE [23] is a UML profile for real-time and embedded systems and SysML [24] is a UML profile designed for systems engineering. Both profiles are endorsed by several vendors and tools, for example Artisan Studio by Artisan Software Tools [25] (support for MARTE is only planned) and Papyrus UML [8].

In conclusion, both metamodeling and UML profiles feature a range of supporting tools.

D. Personal considerations

At this point we would like to inject some personal observations:

- metamodeling gives the user complete freedom over modeling concepts. Conversely, UML profiles only permit additions to the UML metamodel (in the light of our previous criticisms);
- as noted in [5], one distinct drawback of metamodeling is the effort for producing tool support for the DSL, while UML profiles leverage on existing UML tools;
- we must consider the longevity of standard and tools. UML and MOF are clearly well endorsed and this situation is likely to continue. On the other hand, metamodeling tools typically don't use MOF but employ non standard meta-metamodels, the longevity of which is difficult to assess.

III. Problem Two: Proving the Correctness of Transformations

The transformations from PIM to PSM and from PSM to code are obviously central to the essence of the MDE paradigm. They turn the user model into a concrete implementation in accord with choices and directives that reflect the application domain culture and legacy as well as methods defined in the scientific body of knowledge. We must therefore ensure that these trasformations are provably correct.

We say that a transformation from a model A to a model B is *correct* if:

1. everything that holds in A holds also in B. In other words, the semantics of B must contain the semantics of A;
2. things stated in B should not deny things stated in A. In other words, the semantics we add in B should not contradict the semantics contained in A. Note that the trasformation from PSM to code does not add any semantics.

Carrying out such a proof of correctness is a costly endeavor: we must therefore keep its complexity low and its chances of success high. In this respect we wonder what factors impact most on it. We focus our attention on three specific factors, which we suspect to play a central role:

- the number of views used to specify PIM;
- the use of incremental PIM construction, which produces multiple intermediate PSM representations;
- the number of metamodels that underlie PIM views and intermediate PSM.

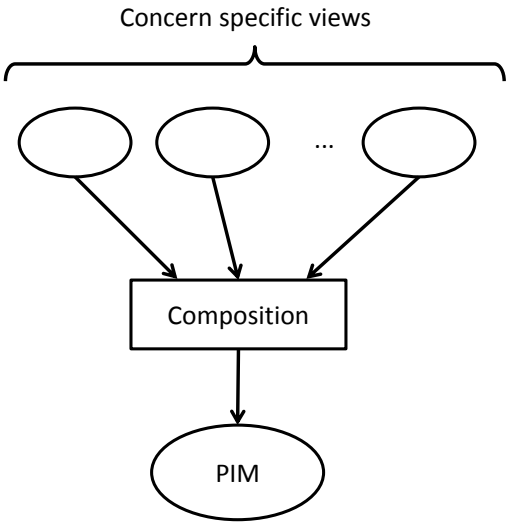

Figure 4. PIM does not necessarily consist of a single view, but it may be composed of several concern specific views, each of them represents a particular concern of the system – for example functional units or deployment of tasks on processors. When we compose the semantics of the views we should obtain a single model representing the system under study.

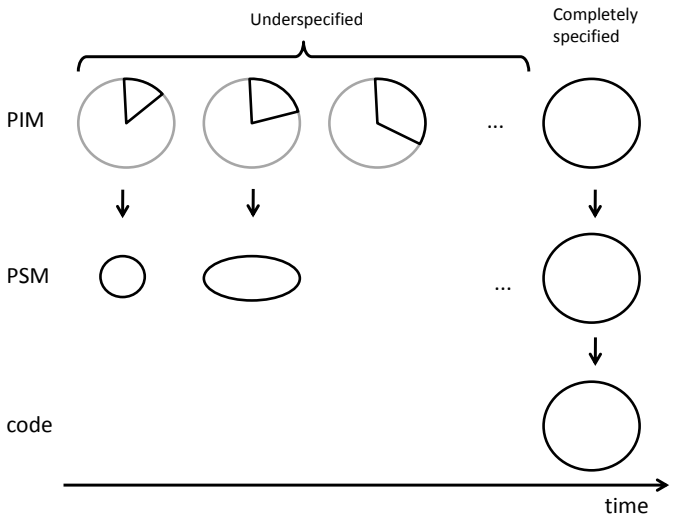

Figure 5. PIM can be constructed incrementally in several iterations; over time we may therefore have a string of underspecified PIM. Under certain preconditions an underspecified PIM may generate a PSM apt for some analyses. PSM derived from underspecified PIM are throwaways and cannot be thought of as increment of a single model – only the PSM generated from completely specified PIM should be kept in order to generate the code for the implementation.

We discuss each of these factors in isolation below.

A. *Creating PIM from concern specific views*

Most real-world software systems are too complex to be specified with a single PIM – creation, comprehension and evolution may become overwhelmingly difficult. We should therefore rather describe it with several models. In this regard, IEEE P42010/D6 [26] (descendant of IEEE 1471 [27]) advocates that the description of the architecture of a system should be made up of a series of *views*, each of which conforms to a *viewpoint* that establishes the concerns of interest (e.g. deployment or functionality) and the ways to address them (e.g. languages and metamodels). This is further acknowledged by SysML [24] that, with reference to IEEE 1471, supports the concepts of views and viewpoints. The more views we admit, the easier and the more accurate the modeling of the overall system.

In order to ensure that the views form in effect a single and coherent system, they should be:

- *composable*: their semantics should not superimpose, or if this happens they should agree on the overlaps;
- *compositional*: there must exist a systematic way to assemble the semantics of concern specific views to obtain the semantics of the PIM as a whole.

In order to trigger the transformation from PIM to PSM, we should prove that the views are composable, carefully minding any overlaps in their semantics. The more views we have, however, the more proofs of correctness we must produce in the PIM space and the more the effort we must expend, which adds to the cost of proving the correctness of the transformation from PIM to PSM.

The cost of determining whether the views are composable in fact depends on the way these views are constructed. There are two main approaches to it, as described in [26]:

- *synthetic approach*: each view is modeled separately with one or more models and later composed with the others;
- *projective approach*: each view is extracted (without any transformation) from a unique underlying model that describes the entire system. Conversely, changes to a view are propagated back to the model.

In the former approach the demonstration of composability of the views implies to show that models underlying them agree on their semantic overlaps. The cost of this activity grows with the number of models – which can be greater than the number of views – and the number of different metamodels used to define them (more on this in section III-C).

Instead in the latter approach this demonstration is straightforward, since views are derived from a single model and this model must be free of contradictions for it to be considered valid. We have still to ensure that views are correctly derived from the model and changes to views are correctly applied to the model, but overall the effort is smaller.

B. *Incremental PIM construction*

Up to this point we have tacitly assumed that we should have a completely specified PIM before we can instigate the transformation of it into the PSM. This assumption sounds reasonable of course, but it carries the implication that iterative feedback cycles – that we may need to improve the goodness of fit of the PIM – can only be triggered on full and complete models. This prerequisite however may be frustrating, because the best use of feedback cycles is to be had when they begin as early as meaningful analysis can be carried out.

We contend that it is desirable to allow generation of PSM from an underspecified PIM, in order that we can obtain early feedback and permit to refine, change or commit design decisions as early as possible (figure 5). While the PIM may be underspecified, therefore, it may still be sufficiently complete for some transformation to PSM to take place for the specific purpose of some specialized analyses. Under this assumption the construction of the PIM becomes incremental. This requirement however has the following implications:

1) things we may leave unspecified in the PIM cannot be arbitrary, but must be determined by the power of the underpinning theory of analysis (which may do away with some detail information) and/or by the analysis procedure (which may permit the use of user estimates in the place of actual values). In other words, it is the PSM and not the PIM to determine what we can omit from the PIM;
2) all the PIM increments can be seen as additions or changes to the same model – although the semantics of these increments can disagree because of the corrections made in response to analysis feedback. On the contrary, the increments of PSM cannot be seen the same way but they should considered as distinct models, since they are generated separately and are tailored for the analyses we run on them. Moreover, except for the PSM corresponding to the completely specified PIM, we can say that all the increments of PSM are "throwaways" – once we have executed analyses on them, they have no further use;
3) the transformation from PIM to PSM becomes more complex – we can in fact regard it as comprised of several transformations, the one for the completely specified PIM (that allows code generation) and the ones for each allowed degree of PIM completeness.
   One important property these transformations should all have is that, given a piece of PIM semantics, all the transformations that address it map it to a piece of PSM that has the same semantics as the one generated by the transformation that targets the completely specified PIM. In other words, we must require that all transformations have a congruent behavior when they operate on the same PIM semantics; if we did not enforce this, then for the various increments of the same PIM we could generate PSM that in fact represent different systems, and the analysis results would in fact be worthless.
   To devise these transformations, it would be best to first define the one that targets the completely specified PIM and next use this as reference to construct the others.

Hence, when we allow incremental PIM construction, the (composite) transformation from PIM to PSM becomes more difficult to prove correct. In fact, to prove that the entire transformation is correct we have to demonstrate that:

1) all the transformations are correct;
2) given a piece of PIM semantics, each transformation maps it to elements in the PSM that have the same semantics as the ones generated by the transformation that works on the completely specified PIM;
3) the preconditions of the composite transformation are correct – that is, we must ensure that for each allowed degree of PIM completeness we trigger the right transformation and only for them.

C. Number of metamodels

We have seen that we may have to deal with a large number of models. Having multiple models implies that we might use more than one language for modeling them and thus we might deal with more than one logical metamodel underneath them. In particular we can have this extreme scenario:

- assuming we are employing the synthetic approach, every model underlying the views in the PIM can have its own metamodel;
- while it is reasonable to assume that every PIM increment uses the same metamodels, we cannot expect this for the generated PSM. Indeed, since in each increment we may perform specialized analyses, we need to express different concerns, for which we may need distinct metamodels.

We talk about *logical metamodel* to suggest that one and the same (mega) physical metamodel may be constructed in a manner that permits multiple logical metamodels to be realized as a specific tailored view of it. For this reason in the following by metamodel we mean logical metamodel.

We saw that the proof of correctness of trasformations requires to handle models' semantics, which is defined through the semantics of their underlying metamodels. We maintain that the number of metamodels to be supported may have a considerable impact on the cost of the proof. If models are specified with different logical metamodels, then before being able to compare their semantics we should perform a *semantic integration* by establishing correspondences between their metamodels' semantics (e.g., this piece of information has the same meaning as that piece of information there; or this relation here is the inverse of that relation there). The cost of this integration seems to grow super linearly with the number of metamodels: if we have n metamodels we have to make a correspondence for each couple of metamodels and then we have $\binom{n}{2} = O(n^2)$ correspondences to make.

Conversely, if models are specified with the same logical metamodel, then the semantics is defined the same way for all models and it is easier to check for overlaps and contradictions. Moreover, if every piece of syntax is attached to precise predefined semantics, the check becomes syntactic.

In fact, UML actually defines a single logical metamodel with several views defined on top of it (the diagrams).

IV. Conclusions

This paper has discussed two problems that are to be faced when we want to leverage application domains and scientific bodies of knowledge in an MDE approach that aims to support the development of correct-by-construction software systems. The problems we posed were the following:

1. how to define modeling languages? Through UML profiles or else through metamodeling? In discussing this question we singled out elements that should guide the decision:

- we criticized the way UML profiles are defined and singled out weak points that hamper the effective use of profiles;
- we noted that the gain of importance of MOF in the context of MDA didn't undermine the importance of UML as a vehicle for defining languages, but rather recognized that UML is apt for metamodeling;
- from an informal survey of tools for creating DSL we reported the observation that both approaches seem to be equally supported.

2. what factors impact most on the complexity of proving model transformations correct? We conjectured that three such factors are:
   - the number of views that make up the PIM. We argued that it may be easier to construct the PIM out of multiple, simpler, concern specific views; in that situation however we must show that the views we use to form the PIM are composable, and the cost of this proof is proportional to the number of views. We also noted that it is desirable to have a single model underlying these views;
   - the incremental construction of PIM. We insisted that we should enable the generation of PSM from under-specified PIM, in order to facilitate specialized forms of analyses useful for instigating (early) feedback cycles. This approach however incurs a more costly proof of correctness. Indeed the transformation becomes a composite one, for which we have to prove that
     1) the transformations inside it are correct;
     2) given the same piece of PIM semantics, all the transformations that may apply to it must behave the same way as the one that targets the completely specified PIM;
     3) each specific transformation is deployed solely when the degree of underspecification of the PIM allows it.
   - the number of metamodels used to specify the views of the PIM and the various increments of the PSM. A proof of correctness requires to deal with semantics; models' semantics are established through metamodels' semantics. Each metamodel defines its semantics in its own way; hence to search for semantic overlaps and spot contradictions we have to make semantic integration between each pair of metamodels. The more metamodels we have, the more semantic integration effort.

REFERENCES

[1] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," *Computer*, vol. 39, no. 2, pp. 25–31, February 2006. [Online]. Available: http://dx.doi.org/10.1109/MC.2006.58

[2] R. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap," in *Future of Software Engineering, 2007. FOSE '07*. IEEE Computer Society, 2007, pp. 37–54. [Online]. Available: http://dx.doi.org/10.1109/FOSE.2007.14

[3] J. Miller, J. Mukerji, and Others, "MDA Guide Version 1.0.1," Object Management Group, Tech. Rep., 2003.

[4] Object and Reference Model Subcommittee of the Architecture Board, "A Proposal for an MDA Foundation Model, ormsc/05-04-01," Object Management Group, Tech. Rep., April 2005. [Online]. Available: http://www.omg.org/cgi-bin/doc?ormsc/05-04-01

[5] I. Weisemoller and A. Schiirr, "A Comparison of Standard Compliant Ways to Define Domain Specific Languages," in *Models in Software Engineering: Workshops and Symposia at Models 2007 Nashville, Tn, USA, September 30-October 5, 2007, Reports and Revised Selected Papers*, ser. Lecture Notes in Computer Science. Springer, 2008, pp. 47–58. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-69073-3_6

[6] "Unified Modeling Language Infrastructure, version 2.2, formal/2009-02-04," standard, Object Management Group, 2009. [Online]. Available: http://www.omg.org/spec/UML/2.2/Infrastructure

[7] "Unified Modeling Language Superstructure, version 2.2, formal/2009-02-02," standard, Object Management Group, 2009. [Online]. Available: http://www.omg.org/spec/UML/2.2/Superstructure

[8] "Papyrus UML," Software vendor tool. [Online]. Available: http://www.papyrusuml.org

[9] "Common Facilities RFP-5: Meta-Object Facility, cf/96-05-02," Object Management Group, June 1996. [Online]. Available: http://www.omg.org/cgi-bin/doc?cf/96-05-02.pdf

[10] "Object Analysis & Design RFP-1, ad/96-05-01," Object Management Group, June 1996. [Online]. Available: http://www.omg.org/cgi-bin/doc?ad/96-05-01.pdf

[11] "UML Proposal to the Object Management Group, version 1.1, ad/97-08-02," September 1997. [Online]. Available: http://www.omg.org/cgi-bin/doc?ad/97-08-02.pdf

[12] "Meta Object Facility (MOF) Core Specification, Version 2.0, formal/06-01-01," standard, 2006. [Online]. Available: http://www.omg.org/spec/MOF/2.0/PDF/

[13] "Request For Proposal: MOF 2.0 Core RFP, ad/2001-11-05," Object Management Group, November 2001. [Online]. Available: http://www.omg.org/cgi-bin/doc?ad/01-11-14.pdf

[14] "Request For Proposal: UML 2.0 Infrastructure, ad/2000-09-01," Object Management Group, September 2000. [Online]. Available: http://www.omg.org/cgi-bin/doc?ad/00-09-01.pdf

[15] K. Duddy, "UML2 must enable a family of languages," *Communications of the ACM*, vol. 45, no. 11, pp. 73–75, November 2002. [Online]. Available: http://dx.doi.org/10.1145/581571.581596

[16] The Eclipse Foundation, "Eclipse Modeling Framework," Software vendor tool. [Online]. Available: http://www.eclipse.org/modeling/emf/

[17] Metacase, "MetaEdit+," Software vendor tool. [Online]. Available: http://www.metacase.com

[18] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi, "The Generic Modeling Environment," in *Workshop on Intelligent Signal Processing*. IEEE, 2001.

[19] C. Amelunxen, A. Königs, T. Rötschke, and A. Schürr, "MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations," in *Model Driven Architecture – Foundations and Applications. Second European Conference, ECMDA-FA 2006, Bilbao, Spain, July 10-13, 2006. Proceedings*, ser. Lecture Notes in Computer Science. Springer, 2006, vol. 4066, pp. 361–375. [Online]. Available: http://dx.doi.org/10.1007/11787044_27

[20] Jetbrains, "Meta Programming Systems," Software vendor tool. [Online]. Available: http://www.jetbrains.com/mps/index.html

[21] Microsoft, "DSL Tools," Software vendor tool. [Online]. Available: http://msdn.microsoft.com/en-us/vsx/cc677256.aspx

[22] ——, "Oslo," Software vendor tool. [Online]. Available: http://msdn.microsoft.com/en-us/oslo/default.aspx

[23] "A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Beta 2, ptc/2008-06-09," standard, Object Management Group, June 2008.

[24] "OMG Systems Modeling Language, Version 1.1, formal/2008-11-02," Standard, November 2008. [Online]. Available: http://www.omg.org/spec/SysML/1.1/PDF/

[25] Artisan Software Tools, "Artisan Studio," Software vendor tool. [Online]. Available: http://www.artisansoftwaretools.com

[26] "ISO/IEC WD4 42010, IEEE P42010/D6," Standard draft, ISO/IEC, IEEE, January 2009. [Online]. Available: http://www.iso-architecture.org/ieee-1471/docs/IEEE-P42010-D6.pdf

[27] M. W. Maier, D. Emery, and R. Hilliard, "Software Architecture: Introducing IEEE Standard 1471," *Computer*, vol. 34, no. 4, pp. 107–109, April 2001. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=917550