Objecting to the Revolution: Model-Based Engineering and the Industry
Root Causes Beyond Classical Research Topics

Gerald Stieglbauer and Igor Rončević
AVL List GmbH, Hans-List-Platz 1, Graz, Austria
{gerald.stieglbauer, igor.roncevic}@avl.com

Keywords: Model-Based Engineering, MBE in Industry, Domain-Specific Languages, DSL Vs. UML, Technology Transfer.

Abstract: By now, Model-Based Engineering (MBE) has a long tradition in academics and research. In contrast to this long tradition, however, adoption of MBE principles in the industry still remain limited. This led to corresponding debates within the modelling community about the root causes of this limited adoption. This paper highlights the importance of these debates and shares the experience gained during many years of technology transfer activity from research to industrial applications. We are presenting two hypotheses beyond classical research topics, for which we have observed in practice that they have the potential to make the adoption of MBE principles in industry more successful. Since these hypotheses are currently based on our observations rather than on scientific research, we want to encourage the modelling community to take the presented aspects into account in their research work.

1 INTRODUCTION

By now, model-based engineering (MBE) has a long tradition in academics and research. In recent years, however, the question to what extend MBE has been already adopted by industry has been raised with an increasing number of discussions within the modelling community. Despite some success stories, the answers to the raised questions were rather disillusioning and were followed by a debate about the reasons and root causes for a limited adoption of MBE in the industry (Selic, 2012). In this debate, the role of standardized general purpose modelling languages (such as UML) is critically reflected and individually designed domain-specific languages (DSLs) became more prominent. In addition, usability and user experience factors were more and more deliberated, especially if current available modelling tools and frameworks are considered (Hutchinson, 2011).

In this paper, we highlight the importance of these debates and share the experience and observations gained during many years of technology transfer activity from research to industrial application (e.g. by participating in large European research programs involving both industry and research institutes). Based on these observations, we derive two generalized hypotheses, which provide - according to our experience - strong indicators to overcome some of the limiting factors and most essential showstoppers for the industrial adoption of MBE. Additionally, we argue that not only technical disciplines but as well non-traditional MBE research disciplines such as psychological and social-cultural aspects as well as empirical engineering should be considered (Whittle, 2013).

Hence, we want to encourage the modelling community to take these aspects into account and to evaluate our hypotheses scientifically to avoid that the industrial adoption of MBE runs the risk of remaining just a niche despite its undeniable potential.

2 REASONS FOR OBJECTING TO THE REVOLUTION

If competing with widely established traditional development methods, the introduction of MBE in industry is often judged as a kind of paradigm change with a negative flavour in form of unpredictable risks that may come along with it. When considering

Stieglbauer, G. and Rončević, I.
Objecting to the Revolution: Model-Based Engineering and the Industry - Root Causes Beyond Classical Research Topics.
DOI: 10.5220/0006216506290639
In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*, pages 629-639
ISBN: 978-989-758-210-3

software, agile methods on a rather low-level of abstraction using (object-oriented) languages such as C++, C# and Java thus remain state-of-the-art in many software development processes, despite the long list of methods and tools proposed by the modelling community. If models are applied, the level of abstraction remains limited, e.g. considering Simulink models to enable programming for non-programmers, using UML models only for drawing class diagrams and code skeleton generation, or using models just for documentation purposes.

In this chapter, we want to re-enact a typical MBE introduction scenario with three groups of stakeholders: the *modelling advocates*, the *management* and the *development team*. This scenario illustrates common reasons for a limited MBE adoption and acts as the basis for further analysis and alternative strategies of introducing MBE in an industrial environment.

If *modelling advocates* speak about essential *MBE ingredients*, they usually mean the active process of designing and introducing an *abstraction layer* in order to reduce the effort of developing complex systems or at least to describe complex relationships in a more understandable manner to facilitate discussions and comprehensibility. Furthermore, they promote clearly defined *semantics* of the models (including a mathematical foundation), which acts as key enabler for effective *automation* features such as *model validation*, *code generation* and *model composability* (Broy, 2010). Last but not least, *separation of concerns* is another essential ingredient for MBE, which promises better structured views depending on a particular aspect of a problem.

In the following sections, we briefly sketch the difficulties with which the modelling advocates usually have to deal when they try to convince both the management and the development team. Then we analyse why the corresponding attempts are likely to fail. In Chapter 3, we present alternative approaches in form of two hypotheses based on practical observations and with the intention to overcome some of the described difficulties.

2.1 The Modelling Advocates and the Management

Despite the very strong conviction of the modelling advocates, convincing the management implies not only to provide good arguments that MBE fulfils its promises, but it means in first place that risks of its introduction are calculable and justifiable. Managers have to make their risk assessment, and with too little input, they reject the request for good reasons. They will argue for instance, why to disturb an established product development process by introducing the risks of a paradigm change, when the company is able to build reasonably well-designed products and sell them with an acceptable margin to its customers. It is difficult to convincingly praise specific long-term effects of MBE, e.g. to emphasize that after an initial phase, productivity and quality will rise. Mentioning this initial phase (usually of unsure length) often amplifies the management's scepticism and the modelling advocates are asked to provide clear evidence if and when the initial efforts are outweighed by the positive impact of MBE.

If no internal success stories are yet existent, the modelling advocates often struggle to prove this evidence caused by a lack of publicly available industrial success stories. Rather rare exceptions such as (Hutchinson, 2011) are unfortunately not credibly outweighing this lack to ultimately convince the management. This comes along as well with a kind of a chicken-or-the-egg problem: who should provide these stories first? For instance, intellectual property issues or competitive considerations restrict the publication of success stories, despite the remaining problem of transferring one success story from one particular domain to another domain or field of application (and promote this transfer understandably and convincingly to the management).

A rather pragmatic position is that profitable corporations just do not experience enough 'pain' to dare a radical and company-wide paradigm change. Consequently, someone could propose to patiently wait until this pain gets over a certain threshold. Certainly this is very unconvincing for the modelling advocates (and the modelling community) and their vision about reducing the *actual* pain and increasing development efficiency and product quality at the same time.

However, if the modelling advocates take this hurdle successfully and get a positive decision from the management, then the even more challenging part just begins: it is up to the modelling advocates to evidently prove that modelling works in the given context and fulfils its promises. Furthermore, since management decisions are usually taken on a larger scale, the modelling advocates have to prove it on that scale as well.

This is usually the point, where companies are internally announcing that they are introducing MBE. Modelling tools are bought and – with the best intentions – the tool department selects the tool for which they can be sure that its features will cover sustainably the state-of-the-art of MBE. Budgets for MBE-centred development projects are released, development staff is strongly motivated to participate in modelling courses, and so on. It is not unusual that this initiatives can last for a longer period (sometimes even years), since the model advocates have convincingly argued that this initial phase would need that time. Consequently, monitoring the MBE introduction process is hard, especially if concrete numbers such as efficiency improvements are concerned.

In parallel to all these efforts, the modelling advocates do not only have to convince the management about the advantages of MBE but they have to get the *development team* on board at the same time. In the next section, we argue that this may be even the tougher job.

2.2 The Modelling Advocates and the Development Team

Even if the management supports the MBE introduction process, additional risks are added by the often underestimated fact that the developer team still can object to the introduction or slow it down significantly for various reasons. This behaviour is not necessarily on purpose, e.g. caused by a general scepticism against MBE approaches. Besides a time-consuming MBE introduction process, however, companies still have to create new products at a high (or even increasing) periodic rate. At the same time, the complexity of making these products is constantly raising, which makes the developers' upcoming deadlines even more demanding. Despite the appreciated long-term perspective of MBE to exactly address this increasing complexity, visiting modelling courses under these short-term demands is considered as an additional burden. This situation gets even worse, if developers are confronted in these courses with complex, feature-rich and poorly integrated modelling tools. This contradiction leads to postponing the adoption of the modelling tool after the deadline, which is obviously just close before the next deadline. What ultimately often overburdens the development team under these circumstances is the demand of abstract thinking or to consider other MBE paradigms such as separation of concerns.

From the developers' point of view, however, the management still demands to square the circle, and periodically wants to see results concerning the MBE integration process. A common reaction is to use modelling tools as documentation tools (e.g. to model system interfaces) with a limited adoption of automation features such as code generation.

However, using modelling just as documentation and even using it as a code skeleton provider is usually a dead end, since there is only a vague or weak connection between the model and the implementation as long as no automatisms such as full code generation are established (and editing fully shifts from the code to the model). The models are sooner than later out of sync with the 'real' implementation (i.e. programming code) and keeping them updated is skipped due to the maintenance effort that is usually considered as 'just' an additional work. The management is usually not in the position to detect that inconsistency, while the developer can argue nevertheless that the corresponding modelling duty has been fulfilled (formally).

After some period, the models are pigeonholed while code-based implementations move forward and hardly anyone can explain the original purpose of the models. Moreover, nobody can tell if they have introduced any efficiency and/or quality improvements, not to speak about the concrete numbers the management is now asking for. Consequently, at some point in time somebody has to report the failure of adopting MBE to the management. It will be argued that modelling was just an additional effort, the maintainability of the models was extremely costly and efficiency and quality improvement is next to zero. This may confirm the management's initial doubts that MBE really works, the topic will be buried for a very long time and the modelling advocates have lost their reputation.

2.3 Root Cause Analysis for a Failed MBE Adoption

In the following, one out of many possible root causes for a failed MBE adoption is briefly sketched. For instance, the modelling advocates argue that models are worth nothing if they are not ultimately connected to the implementation, at best by establishing automatism such as code generation.

At this point, they are confronted with a series of practical problems during the MBE introduction phase. UML is often considered to be applicable to various domains due to its common perception as a general purpose modelling language. In practice, however, the concrete semantics of a model is often implicitly redefined by the individual modeler, who just ignores the (non-formal) UML specification documents. Instead, he or she rather applies an intuitive ad-hoc semantics, which seems to be sufficient, e.g. if the model's main purpose is documentation. However, this ad-hoc semantics is easily misinterpreted by other users, especially from other domains, independently from the fact that adapting the intended semantics to a specific domain often makes sense.

However, if this adaption of the semantics is not done systematically (e.g. by defining a UML profile), the individual interpretability of models causes a series of problems, especially if code generation is considered. First, uncertainty about the model's semantics is intuitively mapped to the functionality of the generated code. To be on the safe side and to get control back about the generated code, self-written code generators are created by the development team in parallel to the standard one, which usually causes maintainability problems.

Second, the costs of creating correct models in traditional modelling tools that fulfill the intended semantics of the (self-written) code generators are considered to be high. Current tools usually lack in features such as modelling guidance that support the creation of such correct models. This is especially true for self-written code generators and gets even worse if the adapted semantics is not well documented. In addition, accustomed developing features such as debuggers are not available in most modelling tools. This usually leads to time-consuming bug fixing on code level again and thus breaks the link to the model, which goes along with the known fatal consequences and finally causes the rejection of MBE by the development team.

3 EVOLUTION INSTEAD OF REVOLUTION

So far, we have sketched a typical scenario based on practical long-term experiences, which argues why the adoption of MBE in industry is often designated to fail. These reasons are not necessarily linked with the theoretical background of MBE but are rather caused by social-cultural and psychological phenomena or just by the fact how companies function (Whittle, 2013).

In the following, we introduce two hypotheses, which attenuate the reasons for a limited adoption of MBE paradigms according to our experience. *Hypothesis A* is related to a strategy called *MBE micro injections*, while *hypothesis B* is about favoring *information re-use* as a key requirement and enabler for MBE. Both hypotheses are derived from industrial use cases as a direct consequence of failed adoption attempts (such as sketched in the previous chapter) and are currently under internal evaluation. Applying the hypotheses to our practice showed first promising results. However, to provide evidence of their general applicability, more research studies are needed. Together with the postulated requirements for modelling methods and tools (reflected in Chapter 4), we would like to handover all these aspects to the modelling community to foster corresponding follow-up research activities.

3.1 Hypothesis *a*: Introduction of MBE Micro Injections

Instead of promoting a company-wide revolution, we rather favor an MBE adopting strategy that emphasizes manageable introduction steps and supports an accurate monitoring process to provide continuous and measurable feedback to the management. We call this strategy *MBE micro injections*. As this term implies, MBE micro injections are limited in size and should have a strong viral effect. They are thus intended to support the convincing process of the development team and thus to reduce the management risks caused by implicit or explicit objects to MBE paradigms.

On the first glance, the term MBE micro injections is in conflict with the argument that MBE will only work, if used very consequently and not just here and there (e.g. using models 'just' for documentation reasons). However, we claim that if these MBE micro injections are well designed and introduced, no essential MBE paradigms need to be violated or omitted at all.

In addition, we suggest to introduce the role of the *MBE micro injection designers* (or *injection designers*, for short). The injection designers collaborate closely with both, the management and the development team, which is outlined in the following.

3.1.1 The Injection Designer and the Management

The injection designers share with the modelling advocates the conviction that MBE has great potential to master complex systems, to speed-up development and to improve the quality of related products. However, instead of focussing just on good arguments, the injection designer aims to proof the advantages of MBE by concrete numbers from the very beginning. In this manner, the injection designer is setting-up an action plan composed of a series of MBE micro injections. Each micro injection is assigned to a short integration sprint and has a measureable output regarding initial efforts, efficiency increase and quality improvements. This allows a constant monitoring of the action plan, which is permanently communicated to the management in order to keep their attention high and to enable the possibility for adaptations, if the numbers are not fulfilling the expectations.

At the same time, MBE micro injections should be designed in a way that they do not disturb the daily business and form up rather a smooth evolution than a shocking revolution. At the end of each integration sprint, the overall solution must not be broken by the introduction of a MBE micro injection, but at the same time, no MBE paradigms are fundamentally violated. The most obvious strategy to achieve this is to install them first in parallel to the traditional solution. This has the advantage of a direct comparison of the competing approaches manifested in concrete numbers reported to management. If the micro injection turns out to be successful it replaces the established solution. If the traditional solution was modified during the evaluation, updating the MBE micro injection accordingly is still manageable due to its limited size.

Furthermore, this strategy attenuates the chicken-or-the-egg problem about the lack of success stories. Each positively evaluated MBE micro injection becomes an in-house success story on a tiny scale indeed but with full access to its details and without IP restrictions. Due to the latter, mapping the positive experiences from one MBE micro injection to another is more straightforward. If MBE micro injections are additionally fulfilling composability criteria (e.g. the same mathematical foundation), the related success stories can be combined step-wise to larger ones in order to ensure scalability. To summarize, these success stories and the perspective of scalability gives the injection designer a much better position when arguing with the management.

3.1.2 The Injection Designer and the Development Team

While modelling advocates may favour the design of abstraction as the most important discipline, the injection designers are not neglecting its importance but set their highest priority on the needs of the development team and afford them to master the introduction of MBE in parallel to the upcoming deadlines. The overall goal is to 'infect' the individual user step-by-step with the MBE paradigms, but not to overload them with a unified modelling philosophy represented by a single, feature-rich tool. Instead, individually designed MBE micro injections need to be highly adaptable and easy to integrate in the existing development environment.

In order to make this strategy viral, the injection designers thus need to know two things at a very detailed level: first, they have to have deep *knowledge about the applied development environment* and second, they have to gain *knowledge about the entered information* (e.g. programming code in case of software development).

Knowledge about the tool environment is important for the following reason: any change to this environment means practically a lot of disturbance facing the next upcoming deadline. It may sound trivial, but is practically a showstopper. For instance, developers are usually not accepting an additional tool, which they have to use in parallel to their well-known development environment. However, developers are much more appreciating new features within their development environment, which can be applied to current development projects instantaneously.

In other words, whatever a MBE micro injection is regarding a concrete use case, it has to be integrated smoothly into the actual development environment and should minimize entry barriers that hinder the user to apply the intended MBE-related features. For instance, instead of just offering a feature-rich UML editor, a feature-reduced sub-editor (e.g. based on a customized UML profile) is directly enabled within the established development environment (e.g. Microsoft Visual Studio). This practically reduces the risk of objection and very much supports the viral approach, since the potentially attracted developer is permanently just one-click away from applying the MBE-related feature.

Of course, the demanding factor here is the smooth integration with a strong focus on the usability of this integration. Consequently, the micro injection designers have to have corresponding skills not only on the technical aspect of a tool integration but as well on user experience issues. User experience, however, does not only comprise usability topics. It includes for instance an *adaptation of MBE standard vocabulary* such as abstraction, modelling, model semantics and model transformation to terms, which are well-known by the user; or in the modelling advocates' words: this adaptation introduces an end-user optimized abstraction layer for the theory of MBE.

User experience topics are also related to the second central aspect of the MBE micro injections concerning the *knowledge about the entered information*. According to our experience, most developers have already an internal meta-model for their approaches in mind. However, it requires special skills to turn an internal, implicit and informal meta-model into an explicit and formal one.

Exactly this task should now be taken over by the injection designer in a very close collaboration activity with the development team. Close collaboration means that the most essential modelling elements are discussed, verified or falsified together with the developers by applying the adapted MBE standard vocabulary in order to avoid confusion. Rapid prototyping facilities lead to a concrete and integrated *MBE prototype* based on a *co-designed abstraction layer*, which are from now on inherent parts of the collaboration. In the following, four *sorts of MBE prototypes* are briefly sketched.

The MBE prototype embraces a *meta-model prototype*, manifested by a concrete meta-model representation (e.g. an EBNF grammar or a list of model constraints) reflecting the co-designed abstraction layer, which is based on the development team's terminology. While the meta-model prototype development is led by the injection designer mostly, corresponding *model prototypes*, which adhere to the meta-model should be created by the developers from the very beginning to capture as many user experience issues as possible during this activity.

To enter such models, *model editor prototypes* have to be provided and continuously improved regarding usability and integration issues. Initial design solutions for model editor prototypes need not be traditional model editors (e.g. a full-featured UML editor). Instead the development of straightforward, but well integrated solutions such as input masks or wizards are often a good starting point. During the meta-model design iterations, more sophisticated model editors potentially enhance or replace this initial approach. In this case, DSLs (textual or graphical or even combined ones) or DSMLs (such as a highly customized UML editor) are considered.

Along the meta-model design iterations, these prototype editors and their features are iteratively refined by the injection designer with regard to their usability. Independent of the usability considerations, these editor prototypes need to be created very fast, usually within days or even better as discussions are going on. To master this challenge, editor generator features greatly produce relief.

If MBE is considered for software development, code generators are absolutely mandatory to ensure the link between the model and its implementation. Instead of implementing handcrafted code generators, which are almost impossible to maintain, corresponding frameworks ease the task of creating *code generator prototypes*. Such frameworks usually comprise generated model parsers (based on the given meta-model or grammar) and support model transformation languages. Similar to the design iterations of the meta-model, however, a developer may not know right from start, how a generalized form of the intended code may look like. Again, the injection designer supports the developer in finding the most appropriate form in several iterations. This task is tremendously simplified, if the applied transformation language is based on code templates, since developers are then able to provide their input in a language they already know. In the best case, they are even able to migrate legacy code to these templates to avoid a full reinvention of existing approaches.

3.1.3 Psychological and Social-cultural Aspects of MBE Micro Injections

It is essential that all four sorts of MBE micro injection prototypes are developed in short and agile sprints that minimize the disturbance of the developer. First approaches are more or less designed by the injection designer in the lead, but are rather applied in parallel to the traditional methods of the development team in order to compare and proof the benefits of the MBE approach. The comparison comprises a concrete performance benchmark in terms of efficiency and quality improvement.

Before reporting immediately to the management, the benchmark serves primarily to *establish trust* between the injection designer and the development team. The development team has furthermore the possibility to evaluate the methods and tools on their own with support of the injection designer to avoid misconceptions. As cooperation develops during the design iterations, the development team is encouraged to take over more and more tasks from the injections designer, such as modifying the model or editing the model transformation rules.

Another psychological effect is the individual perception among the development team members that the introduction of high-level abstraction limits their solution space compared to the use of low-level abstractions. This perception is not so much caused by vanity issues but is rather related to the fear of *losing control* about the final implementation (e.g. in case of code generation). This effect is reduced, if the developers know where to put the right action and what it takes to do corresponding adaptations to avoid this problem (and not to endanger next week's deadline). Thus not only trust between the development team and the injection designers is mandatory but to give the development team the perspective to *retain control* and to gain the ability to do essential adaptations even without the help of the injection designers. The use of template-based model transformation approaches for instance, where familiar programming languages are embedded, supports this strategy. As an important side effect, this is also mandatory for the viral aspects of the MDE micro injections: without handing over some of the MDE skills back to the development team adopting MDE will not be sustainable.

Having this in mind, the injection designers motivate the development team to present the performance benchmarks to the management to give them the opportunity to sell the results as their achievement, while the injection designer remains in the background. This should amplify the viral effect of MBE micro injections within the development team and complements the activities of the injection designer to convince the management.

In some cases, however, the introduction of abstractions may also affect the development team's pride of making their original solution less genius (i.e. less complex), due to the fact that it can be generated (a kind of substitution of their skills by an automated process). In addition, if their original solution is representable in a much simpler way, i.e. by a model, and thus the solution can be understood now as well by less educated people, there is a fear that the role of individual members of the development team becomes questionable.

All these aspects should be considered by the injection designer and corresponding social skills are mandatory here to attenuate the negative effects and possible fears. The best way to cure a wounded pride remains the involvement of the development team and not release them from responsibility. Much more could be said about optimized training conditions and the responsibility of the management to promote an open attitude by relieving the development team from some of the pressure. However, being complete here goes beyond the scope of this paper. What the injection designer can do is to convince the members of the developer team that MDE is not limiting their skills but is rather a catalysts of their abilities to master even more complex solutions in the future.

3.2 Hypothesis *B*: Information Re-use as an Enabler of MBE Adoption

For our *second hypothesis B*, we want to present an application field of MBE, which is usually not nominated as a first-class driver for MBE. Instead, issues such as the reduction of complexity by the introduction of abstraction or the principle of separation of concerns are usually designated as primary reasons. However, finding and defining the right abstraction layer in order to discover a global minimum of complexity of a sophisticated system is not a straightforward task. Instead, this task depends on certain skills, which are usually covered only by a perfectly trained and educated *MBE designer*.

In practice, this noble goal is thus sometimes beyond the reachability and too high expectations even increase the risk of failure. In addition, aiming at a global minimum of complexity contradicts with the postulated *hypothesis A* – the MBE micro injections – which rather counts on the composition of local improvements.

We claim, however, that there are some low hanging fruits, which are much more likely to be harvested and implicate a manageable form of abstraction by a quite straightforward paradigm: the paradigm of *information re-use*.

Information re-use can denote various things. It could mean for instance that code fragments of one project have to be re-used within another project with a related purpose, eventually in a slightly modified manner. In this case, the MBE designer would introduce a certain layer that abstracts the similarities away and introduce a meta-model, which focus on the differences only, and generate the remaining, i.e. schematic repetitive code automatically. For someone, who has to re-use some code fragments in his or her project, modelling just the differences is much simpler than modifying the original code source again and again. We think that applying this principle leads to a clear methodology for defining abstraction layers, but still make high demands on the person who has to find the similarities or common semantics of different code fragments (especially in case of legacy code).

However, we claim that a slightly modified variant of this principle could be applied more straightforward, especially in large companies (for which the introduction of MBE may be most demanding). Large companies are usually comprised of many departments. Here the situation is common, that communication works well within a department, but at the same time the efficiency of information share and re-use across several departments is limited.

Usually, when information is shared between departments, e.g. between a software department, a documentation department and a service department, classical communication methods such as e-mail or shared folders with informal documents (e.g. presentation slides, textual documents) are common. Sometimes, the situation is often worse since it is not ensured that shared information originates from the responsible author. Instead, information is re-phrased and during this process it gets lost or is adulterated.

To overcome this situation, a primary source of any kind of information needs to be established. Let's call this source the *single source of truth* (SSoT). An SSoT, however, does not mean to install a single physical source (e.g. a huge database) that covers everything that has to be shared. This could take years and such a database is likely to be outdated by the time it comes to life. Instead it adheres to an adoption of the MBE principle of *separation of concerns*: if an SSoT *X* established within domain D_X is based on information of another SSoT *Y* within domain D_Y (and potentially vice versa), information referencing is used instead of information copying or translating. Or in other words: SSoT *X* is enriching the information of SSoT *Y* by further aspects needed in domain D_X (and potentially vice versa).

We postulate, when optimizing information exchange in the described manner, the discovery and the establishment of these SSoTs can go hand in hand with very effective MBE approaches. A simple rule here is that every SSoT has a single author *A* (e.g. a software developer) and a series of consumer C_n (e.g. service engineers or technical authors), which are using and processing the information in their specific domains. Consequently, only the author *A* is able to edit the content, while the consumers C_n have read-only access. The sum of all shared data items for a specific SSoT defines the overall abstraction of the SSoT, which is translated by the MBE designer to a machine-readable *SSoT meta-model*. This approach intends to ensure that the amount of shared data between the stakeholders is reduced to a minimum and relieves the consumers C_n from time-intensive searches in overloaded and potentially outdated documents.

On top of the SSoT meta-model, the most suitable model representation in terms of an optimized user experience has to be designed for the author *A* and for the consumers C_n. Classical modelling approaches (such as UML) suggesting a standardized modelling language, which can be understood by both, the author *A* and the consumers C_n and sharing information means to share the same model representation. In many situations, this makes sense and increases the cross-domain knowledge since the stakeholders are 'speaking' the same (model) language and are modifying the same model, which improves cross-department collaboration.

However, defining a universal representation for various stakeholders from different domains can only be a compromise in terms of the user experience. In addition, learning the language means initial effort, which is sometime hard to acquire in practice. Furthermore, a stakeholder from one domain gets access to details of another domain with no further use. At the same time, accessing the relevant, domain-specific data becomes harder.

The SSoT principle as defined in this section, relax this dilemma, since only a single stakeholder has to edit the content of a particular SSoT *X*. Due to the principles of separation of concerns, other stakeholders are editing other SSoTs and are just referring to SSoT *X*. Consequently, different model representations can be designed for various stakeholders for the SSoT *X*. For instance, a textual DSL editor E_X is designed for the author *A*, who may be a software developer. However, considering the read-only model view V_X for the consumers C_n, e.g. a technical author C_T and a service engineer C_S, the corresponding model representation might be something completely different (e.g. a selection dialog of relevant model elements). C_T and C_S may not even share the same view. Instead, separate views V_T and V_S show only that information to C_T and C_S to which they have to refer in their domains D_T and D_S. Both model views V_T and V_S and the model editor E_X of the author *A* for the SSoT *X* can now be optimized in terms of user experience independently from each other.

If we combine hypothesis *B* with hypothesis *A*, i.e. the introduction of MBE micro injections, the following aims become tangible: first, finding the right abstraction layer is reduced to the method of identifying the relevant information that is exchanged between two stakeholders. This is achieved by a series of interviews, where the MBE designer is questioning the stakeholders regarding the essential information they are currently obtaining from other stakeholders via traditional methods. Second, the MBE designer identifies where the shared information originates and who is the author. Based on both inputs, the MBE designer defines the list of SSoTs according to the principle of separation of concerns and creates a meta-model for each SSoT. Finally, it depends on the context if these SSoTs are physically combined to a single database or not.

Similar to the MBE micro injections, this process of SSoT establishment can be done in small introduction steps. This improves already the status quo if a certain kind of data set, which was formerly exchanged by traditional methods, is now accessible via an SSoT. Of course, a seamless integration of the corresponding user front-ends (i.e. model editors or viewers) is essential. Once this is successfully established, the list of SSoTs can grow continuously in size and number. For every new SSoT a measurable before-after-comparison should be possible, e.g. by measuring the efficiency increase of the information exchange between two stakeholders.

This comparison is reported periodically to the management and thus enables continuous monitoring of the SSoT introduction process.

If the vision of this approach and the claim of our hypotheses hold, then the involved stakeholders are more and more infected with the ideas of MBE and the SSoT paradigm. They have learned what is meant by abstraction layers and are introducing further layers by their own. To give just one example, software developers may adopt this global approach of information re-use locally as indicated before: they introduce transformation rules based on code templates to generate standardized implementations and to enable code re-use. If a software bug has been detected, it can be fixed within the standardized code template and thus positively impact all applications that are based on that template at once.

Thinking in abstractions is becoming more and more common and the viral approach is continuously growing, or in other words: once the way of thinking of the development team has been successfully shifted to a higher level of abstraction, it will remain there and will infect other people autonomously.

4 STATUS-QUO AND DERIVED REQUIREMENTS FOR MBE TOOLS AND FRAMEWORKS

In this chapter, we want to summarize the requirements for MBE tools and frameworks, which are especially related to the hypotheses presented in this paper, i.e. the *MBE micro injections* and *information re-use*. In addition, we want to enhance these requirements with further details and confront them with our experiences with currently available modelling tools and frameworks.

Out of our experience, the definition of MBE micro injections contrasts the traditional introduction of MBE by feature-rich modelling tools as an all-in-one solution. Instead, *customizability*, *adaptability* and *composability* as well as the ability of a *straightforward integration* into existing development environments are considered to be the *key requirements* for MBE micro injections. All requirements come along with a series of additional *user experience requirements*, such as *tool usability*, *comprehensibility*, the support for *user guidance* and *rapid prototyping*, adequate *learning curves, collaborative modelling* and *trustworthiness* for the applied methods, tools and frameworks.

Especially *customizability*, *adaptability* and the *user experience requirements* are strongly related with actual discussions within the modelling community about the pros and cons of general purpose modelling languages (such as UML) and domain specific languages (DSLs). Considering these discussions, we are convinced that DSLs are most suitable for the proposed MBE micro injections approach and are targeting best in fulfilling the mentioned requirements.

Current trends (e.g. within the Eclipse modelling community) aim as well in this direction. For instance, the Papyrus industrial consortium (Bordeleau, 2014) is denoting customized UML editors and extended support for UML profiling as a so-called DSML and is considering the Papyrus modelling tool rather as a customizable UML editing framework than a full-featured UML standalone tool. *User experience* is targeted by enabling *collaborative modelling* features, for instance as achieved with the seamless integration of EMF Compare and Egit (Langer, 2015). *Composability* and the *ability for a straightforward integration* is ensured by an underlying unified data layer called Ecore and the intended integration of fUML in Papyrus is intended to bring formal semantics for UML into life (Guermazi, 2015).

Pure DSL frameworks and Eclipse plugins such as Xtext, Xpand and Sirius cover these requirements and additionally capture the need for *rapid prototyping*: Xtext, for instance, supports editor and parser generation while Sirius enables the design of a graphical model language with just a few clicks. Xpand allows straightforward access to the generated model parser and has support for a template-based code generation approach with an *adequate learning curve*. Generated DSL editors defined in Xtext are covering *usability aspects* by providing features such as syntax highlighting, automatic code completion, tool tips and quick fixes more or less out-of-the-box. These features are belonging as well to the demand of *user guidance*, which gives the user immediate feedback during modelling. In case of Xtext, some of this feedback is embedded in the DSL grammar definition and thus part of the MBE design. Related to our hypotheses, this enables the MBE designer to adapt the modelling feedback to the needs of the development team and thus greatly helps to encourage the development team to take over more and more modelling tasks from the MBE designer.

Despite of all these promising developments within the Eclipse community, many limiting factors still remain when integrating these frameworks within an industrial environment. In practice, modelling frameworks often lack in maturity and the MDE designer has to be a framework expert with deep-insights about the framework internals in order to apply adaptions and customizations with a reasonable performance. The documentation usually fundamentally lacks of many important details, interfaces are unclear or redundant, and reasons for framework malfunctions are hard to detect. All these factors contradict the requirements of *enabling rapid prototyping* and *trustworthiness* of the applied solutions. This especially affects the close collaboration between the MBE designer and the development team in a negative way and raises the probability to object to the MBE adoption process.

Many *user experience* aspects are still underrepresented but must be treated as first class requirements from our point of view. Compared to off-the-shelf IDEs for traditional programming languages, *user guidance* features have by far not reached the same maturity level. Despite the better situation for DSLs, traditional modelling tools usually significantly lack in *user guidance*: hardly anything is indicating to the users that they are doing something right or wrong, nor any suggestions for a certain modelling context are provided. Besides increasing the probability of rejection, the lack of *user guidance* is a reason for individual interpretation of model semantics with the known fatal consequences.

Defining a formal meta-model semantics underneath (such aimed by fUML), only partially solves the problem of misinterpretation. The user still has to understand the formal specification, but in practice many formalism representations are considered to be rather discouraging due to their mathematical notation and the (subjectively) implied poor *comprehensibility*. A promising approach here could be to hide the formalism from the end user, but instead derive *user guidance* features directly from these formalisms.

On the other hand, formalized semantics are the basis for the requirement of *composability* (Broy, 2010), which we consider as essential regarding our hypotheses of MBE micro injections: if the applied injections remain just local islands and are hard to combine in form of a step-wise integration process, the overall MBE adoption strategy will fail.

Finally, another potential showstopper has to do with the dominance of the Eclipse community regarding modelling frameworks. On the one hand, a kind of monopolism is even an advantage here, since the Eclipse frameworks are on the way to become the de-facto standard for modelling tools and frameworks, which makes tool decision and *integration* much easier und supports the requirement of *trustworthiness* in terms of long-term tool availability, especially due to its open-source philosophy (Bordeleau, 2014). On the other hand, it still remains a challenge to integrate traditional of-the-shelf development tools outside the world of Eclipse-based modelling frameworks (e.g. Microsoft Visual Studio). The existence and widespread use of these development tools, however, cannot be discussed away. It is practically impossible to migrate extensive development projects with a significant amount of legacy from one platform to another. However, it would be unfortunate to exclude half of the potential modelling advocates just because their companies are not using Eclipse-based development tools. Thus corresponding platform bridges are fundamental and need to be much more promoted.

5 CONCLUSIONS

We have observed that the two hypotheses presented in this paper have the potential to make the adoption of MBE principles in industry more successful. However, these hypotheses are based on long-term experiences within an industrial environment rather than on scientific research. Thus we want to encourage the scientific modelling community to put some attention on them in form of further evaluations.

In addition, the two hypotheses are related to a series of requirements for MBE methods, tools and frameworks. We have acknowledged corresponding trends in the modelling community to address requirements such as customizability and user experience issues, which are currently mostly reflected by DSL approaches. From an industry point of view, however, many of the mentioned requirements are still not manifested enough in MBE tools and frameworks. Consequently, we would appreciate if the modelling framework and tool community put even more focus on these topics.

REFERENCES

Bordeleau, F., 2014. Model-based engineering: A new era based on Papyrus and open source tooling. In *Proceedings of the 1st Workshop on Open Source Software for Model driven Engineering co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages*, pp. 2-8.

Broy, M., Feilkas, M., Herrmannsdoerfer, M., Merenda, St., Ratiu, D., 2010. Seamless model-based development: from isolated tools to integrated model engineering environments. In *Proceedings of the IEEE*, pp. 526-545.

Guermazi, S., Tatibouet, J., Cuccuru, A., Dhouib, S., Gérard, S., Seidewitz, E., 2015. Executable modeling with fUML and Alf in Papyrus: Tooling and Experiments. In *Proceedings of the 1st International Workshop on Executable Modeling co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems*, pp. 3-8.

Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, St., 2011. Empirical assessment of MDE in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, pp. 471-480.

Langer, Ph., Koegel, M., 2015. Integrating open-source modeling projects: Collaborative modeling with Papyrus and EMF Compare. In *Proceedings of the International Workshop on OpenSource Software for Model Driven Engineering co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems*, pp. 30-37.

Selic, B., 2012. What will it take? A view on adoption of model-based methods in practise. In *Software and System Modeling*, pp. 513-526.

Weigert, T., Weil, F., 2006. Practical experiences in using model-driven engineering to develop trustworthy computing systems. In *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, pp. 208-217.

Whittle, J., Hutchinson, J., Rouncefield, M., Burden, H., Heldal, R., 2013. Industrial adoption of model-driven engineering: Are the tools really the problem? In *Model-Driven Engineering Languages and Systems – 16th International Conference*, pp. 1-17.