Response to the reviews – ECMFA 13

Paper 14: “Characterization of Adaptable Interpreted-DSML”

Eric Cariou, Olivier le Goaer, Franck Barbier and Samson Pierre

This document presents a response to the reviews of paper #14 submitted to ECFMA 13. In a first part, we summarize the main changes of the paper. Then, we answer to each point of the reviews and explain how they have been taken into account for modifying and enhancing the paper. Similar critics or questions have been raised by several reviewers. For avoiding the duplication of answers, they have been numbered. This allows us to reference the same answers when reviewers' concerns are the same.

Overview of paper modifications

Four main modifications have been made in the paper content, based on the remarks and questions of the reviewers:

1. A clarification of the introduction concerning the comparison of the approaches (b) and (c) of Figure 1 (direct model execution adaptation vs applying models@run.time on model execution). In complement, the paper's conclusion presents a perspective on the detailed study of pros and cons of each approach.
2. An addition of a sub-section to section 2 for explaining the particular nature of executable models as most of the reviewers pointed out that our models seem not to respect the abstraction principles of modeling.
3. The suppression of all parts dealing with adaptation scope (domain vs business). Indeed, our initial dissertation was insufficient and unclear. Moreover, this point does not have any impact on the proposed characterization.
4. Addition of a related work section.

Meta-review

The reviewers agree that the paper tackles important, interesting and non-trivial issue. However, due to several problems in the presentation, we can only ****conditionally**** *accept this paper. The authors should send to the PC chair a letter that summarizes how they have addressed the reviewer comments.*

At least the following two points should be addressed thoroughly:
(1) a discussions of limitations and drawbacks (cfr., R2),
(2) a "Related Work" section where the authors review the previous works, making their contribution (compared to them) explicit and clear (cfr., R1).

Only if these suggested points for improvement have been tackled we can accept this paper to the Foundations track. Moreover, we encourage the authors to go beyond these specific two points for improvement, considering that various other valuable suggestions have been made by the reviewers.

A1. Concerning the first point, about limitations and drawbacks, we have only general-purpose answers complementing those presented in the paper. Actually, one of our main perspectives is to compare the adaptation of an executed model either by directly adapting it or by applying common models@runtime adaptation techniques on it (the (b) and (c) variants of Figure 1). This perspective was cited in the MRT12 workshop paper [5], but it has not been recalled in the initial version of this paper. Both approaches have pros and cons. Common adaptation techniques require to defining and implementing a causal and reflective link between the system and the model. On the other side, direct adaptation can lead to a too complex adaptation management when mixing together adaptation and execution. For this reason, we consider that the two approaches are complementary and not opposed. Depending on the context, one approach will be more suited than the other one. We focus on this paper on the direct adaptation because this is a more novel way to be explored concerning the adaptation of executed models. Indeed, using models@runtime for adaptation has been widely studied and its application to model execution will mainly lead to adapt already existing adaptation works to this particular context.

Proposed corrections: we have extended the introduction to be more precise on the complementarity of the two approaches and why we stress the direct adaptation of model execution. In the paper's revised conclusion, we have recalled the perspective of studying the concrete differences between the two approaches and their pros and cons. Previous to this study, we will enhance our MOCAS platform for getting a platform realizing adaptation for model execution through models@runtime principles.

MOCAS is an adaptation platform for executed models. More precisely, MOCAS is based on an a UML agent state machine observing an executed business UML state machine and requiring changes on its current state or structural content. In that way, MOCAS seems to be an example of an adaptation platform for model execution through models@runtime principles. The problem is that the adaptation logic and the model execution are strongly mixed-up within the execution engine leading to not having a clear separation of the adaptation from the running system. The enhancement of MOCAS will consist in realizing this separation. MOCAS was not cited on the initial version of the paper because we were only focusing on the direct model execution adaptation.

Finally, as a more general answer to this point, our paper is about defining a characterization of adaptable interpreted-DSML based on the extension of the well-known characterization of interpreted-DSML. We propose in the paper a general and conceptual framework, at a high-level of discussion, even if we give a concrete example as an illustration of the general ideas. As a consequence, some of the remarks or questions of the reviewers on specific points are borderline when compared to the deep paper's topic. For instance, precisely stating the drawbacks of the direct adaptation of the executed model, how to ensure the validity of the adaptation or how precisely the engine processes both execution and adaptation. Each of these points has to be – and will be in the future – developed in a dedicated paper.

A2. Concerning the second point, about the related work, there is already related work in the paper. The section 2, giving the characterization of i-DSML, contains only related work. Indeed, we here summarize the well-known and consensual definition of model execution. We have modified the introduction of the section 2 to clearly state that this section is summarizing the cited works at the beginning of the section [2,3,6,7,8,9,12,14]. We do not present something new or different compared to these works. However, we do not want to change the name of this section in "related work". Naming it as the "characterization of interpreted DSML" is important for the structure of the paper since the next section shows an extension of this characterization by adding adaptation to it.

The second part of the paper where there is some survey on related work is the beginning of the section 3. We explained that the only paper we found, characterizing adaptation of model execution like us, is [12]. From our point of view, this paper was the only one that we could study in a "related work" section, but we do not really think that a 10 lines length related work section may bring out added value. This is why it was been added in the beginning of the section 3. However, this pollutes this section with unrelated material compared with the goal of the section. So, as proposed by the reviewers, in the revised paper, we have added a dedicated related work section just before the conclusion. The section also extends the scope of the related work part by studying paper discussing model execution adaptation, either by using direct adaptation of the executed model (case (b) of Figure 1) or by applying common reflexive adaptation techniques on a model execution (case (c)) such as MOCAS is doing it partially. However, we were not able to find others papers than ours on these subjects.

With these modifications and clarifications, we think we have a better related work survey in the corrected paper.

Review 1

This paper describes the main concepts of interpreted Domain Specific Modeling Languages (i-DSML) for enabling executable models in a MDE context. This description is extended to characterize adaptable i-DSML, where models may not only be executed, but also adapted at runtime.

As mentioned by the authors, software adaptation is becoming more and more important, and MDE offers a great foundation to contribute to this topic. The authors address an important issue, trying to establish the essentials of adaptable i-DSML. They provide a case study of a train example, implemented in Kermeta, which facilitates the understanding of the proposal. The overall structure of the paper is clear and easy to follow.

Some remarks:

The introduction presents three possible approaches for implementing the adaptation loop (summarized in figure 1). The authors state some disadvantages for the first one (i.e., it requires to maintain the consistency between the system and the model) and the second one (i.e., it requires an increased complexity, as the model must contain all the information required for its execution and adaptation). Finally, they present a third approach that seems to be the most balanced and appropriate one, as it tries to alleviate the disadvantages of the two previous ones. This is quite confusing as the authors focus the paper on the second approach, without explaining whether their contribution to this approach addresses its disadvantages, or how it could be applied to the third one.

A3. On this point, our introduction was not clear. As written above in A1 (Meta-review section), we have modified the paper's introduction for clarifying this point. Namely, the case (c) of Figure 1 is just a restricted version of the case (a) where the system is a model execution system instead of any particular system. Then, it has the same drawbacks than (a). The choice for realizing the adaptation in case of model execution is between, directly modifying the executed model (the approach presented in the paper, case (b)) or applying classic models@run.time techniques (case (c) being a particular version of (a)).

As mentioned by the authors, defining executable models is not a novel idea. Thus, I miss a "Related Work" section where the authors review the previous works, making their contribution (compared to them) explicit and clear.

A4. As written above in A2 (Meta-review section), section 2, totally, is a related work section about model execution. In this section, we do not propose anything particular or new definition of model execution but summarize the consensual definition of model execution. Moreover, a related work section has been added before the conclusion.

Other important concerns about the proposal presented in this paper are listed next:

(1) At the beginning of section 2.2, the authors state that a meta-model deals with the abstract syntax of the language, and the transformation (in case of code generation) or the execution engine (in case of model interpretation) captures the semantics. I totally agree with this foundation. However, figure 5 seems to be contradictory, as semantics are included as part of the meta-model. Shouldn't they be part of the engine? If the authors mean that the meta-model provides the designers with some concepts for describing the execution semantics of the models, this would imply that designers will need to define, for each model, how it behaves. However, execution semantics is common to all the models conforming to the same meta-model. For instance, given a modeling language for specifying state-machines, designers do not need to specify how transitions are executed in each particular state machine. Conversely, once the execution semantics of a transition is defined, all the transitions in all the state-machine models will behave identically. Actually, this is something noticed by the authors in section 3.1: "execution semantics deals with a domain level as an execution engine is supposed to be able to execute any model conforming to the i-DSML". What seems more reasonable is that the meta-model is associated (association without containment) with the specification of the execution semantics (e.g., through operational semantics), which is then implemented by the interpreter.

A5. Yes, you are right. We have corrected the figures 2 and 5 as the containment association between i-DSML and Execution Semantics is not the right kind of relation. We have also added an implementation association between the Execution Engine and Execution Semantics.

However, this approach does not fit well with adaptation semantics. Regarding the adaptation scope defined in section 3.1, adaptation can be application dependent (what the authors call business level) and, thus, the associated semantics should be specified at the model level and not at the meta-model level.

A6. There are, for all the reviewers, a lot of confusion or expected clarification about the adaptation scope (domain vs business). This means that our explanations are clearly insufficient and incomprehensible. The problem is that stating what concretely these two levels are, is a too long conceptual dissertation. However, considering an adaptation at the business or domain level is a secondary point; it does not have any impact on our adaptable i-DSML characterization. Business and domain adaptation checking or actions are managed in the same way. So, we have decided to remove from the paper all the points around the adaptation scope.

But, here is a short concrete explanation: The only difference between the two levels of adaptations is that business ones are based on specific business values. For instance, if an event `evt` appeared, checking `evt.name = 'Red'` is at business level because it is based on the particular Red event that is dedicated to a given business context whereas checking `myEvents -> exists (e | e.name = evt.name)` is at domain level since no particular business values are used.

To conclude on the adaptation scope, we plan to write a paper dedicated to this topic in which concrete examples will help in understanding the concept.

(2) It is strange that the adaptation semantics extends the execution semantics, as they have different natures. For instance, the authors consider the adaptation operation at two levels (business and domain), whereas execution semantics is commonly placed at a domain level.

A7. Again, the adaptation scope is a secondary point and has no impact on any part of our characterization. As written in the paper and recalled in conclusion, one of the main differences between an adaptation and an execution action is its intention: an execution action deals with a nominal behavior whereas an adaptation is concerned by uncommon or abnormal situations. Nonetheless, both deal with behavior and are processed by the execution engine.

(3) Concerning the dynamic part of the meta-model, the authors do not justify why models need to be self-contained. The dynamic information can be totally located in the interpreter, in the model or balanced between both. I understand that each solution have pros and cons. For example, representing the run-time state in the model may increase complexity, polluting the meta-model with many details not relevant at design-time. Conversely, a model contains a complete snapshot of the execution, which can be helpful for debugging tasks.

A8. Yes, debugging is made easier with a self-contained model. We give some other similar reasons at the end of the section 2.2 on execution engines. However, a more general answer is that executable models are particular models. Executable models aim at being substituted to code. Conceptually, there are at the same level than code: at the lowest one, far away from abstract design models. So, they must contain a lot of details. Moreover, their intrinsic nature is to be executable; it is then logical that the model contains elements related to execution. All the works we cited at the beginning of section 2 are research considering that an executable model contains its current state representation during execution. Some of these research works go beyond by considering that the model can contain its complete execution trace (i.e. the sequence built with the current state of the model after each execution step).

Owing to these remarks on the nature of executable models are recurrent, we added a new sub-section on this point: "Section 2.3 Self-contained Executable Models". Moreover, the sub-section 2.2 has been extended for describing the particular cases when a model does not store its current state (typically when an existing meta-model has not been designed for that, such as all dynamic UML diagrams).

(4) Extending the meta-model for including adaptation specifics goes against the principle of "Separation of concerns", which promotes the separation between the adaptation logic and the application logic. This principle providers many benefits (maintainability, extensibility,...) and has become a foundation for adaptive systems.

A9. There are several reasons for including adaptation elements in the model. The first one is explained in introduction: we want to avoid the drawback of maintaining a causal and reflective link between the system and a model representing it; for this reason we investigate the direct adaptation of the executed model. This requires for the model to contain all the necessary information, including adaptation elements.

The second reason is that we want to remain in the "spirit" of executable models that are by nature self-contained. Possibly, we could have two models at runtime: The executable model and another one containing adaptation elements (however, the required cooperation may create some burden). For our train example, each state and each event are associated with a speed property. It would be possible to put all these properties into another model but each property has to reference its source element in the train model. So, in such a case, this separation would only be logical and would raise some problems of consistency between the train model and its property model. Moreover, such speed properties must be valued by the train designer; they can be considered as belonging to the business content of the model.

The third and more general reason is that an adaptable executable model is at the lowest level of the development process. So, at this level it can contain a lot of mixed concerns but the development process leading to this model can be based on the principles of separation of concerns. We can imagine starting from a business model (and its associated meta-model), and, after a series of transformations and refinements, that the business model (and its associated meta-model) is augmented with elements related to execution and adaptation. We can, for instance, imagine that any i-DSML can be automatically modified for adding to it the adaptation part of our state machine meta-model (*AdaptableElement*, *Property* …). However, the goal of our paper is to characterize what is an adaptable i-DSLM and not to describe some development processes that can define them starting from a high design level (even if such processes are of course interesting to study and develop). Finally, concerning the separation of concern, the goal of the DSML of orchestration presented in conclusion is also to clearly separate the adaptation logic from the execution one and how they are weaved (see next answer A10).

Finally, the way the adaptation logic is expressed is not completely clear. Both the adaptation checking and the adaptation actions are directly coded in the interpreter and not modeled. Is it necessary to modify the interpreter each time a model needs to include a specific adaptation action? Does the proposed approach deal with self-adaptation? At the moment, the user decides which adaptations are going to be applied in the train example. How could the work be extended for applying an optimization-based adaptation as the one proposed by Fleurey (reference [10])?

A10. Yes, self-adaptation is possible. Actually, we provided an example of self-adaptation: when the White signal is detected by the train, thanks to its associated properties, the model is automatically modified to take into account this unknown signal and without having any "hard-coded adaptation" dedicated to this particular event (or any interaction with the user). Concerning the work of Fleurey, the main idea is that their DSML (defining properties, values and associated rules) can be used for being the adaptation part of a meta-model. Our engine will be responsible for managing these elements during the execution and taking adaptation decisions based on these elements.

Finally, the adaptation logic is currently implemented in an ad-hoc way in our prototypes. As explained shortly in the paragraph "Adaptation Checking" of section 3.3, the adaptation checking can be realized through pre-conditions of an execution operation. If these pre-conditions are satisfied then the execution operation is performed else an adaptation action occurs. However, one of our main perspectives is to define a DSML that will enable the orchestration and the weaving between the adaptation checking and actions on one side, and the execution operations on the other side. The engine will use an orchestration model for executing and adapting a model. This perspective has been more precisely explained in the conclusion.

Review 2

This paper presents the concept of adaptable, interpreted (i.e., directly executed) DSMLs. It shows how they can be used to perform high-level programming with 'self-modifying code' (my words :-)).

The paper is very well written and presents an interesting concept. It is perhaps more of a vision paper than a full conference paper in that it doesn't seem to contain much of an evaluation etc.

I am happy to propose acceptance of the paper. However, there are also some things which I would have liked to see in the paper:

- It would be nice to give a more explicit treatment of the interaction between interpreter and adaptation opportunities. You hint at the fact that the possible adaptations are limited by the intepretation engine, but it would have been nice to see a more explicit treatment of this in the examples and, perhaps, even in form of a theory.

A11. This question is similar to one of the first reviewer. See the second part of the answer A10 above. We plan to define a DSML that will enable the orchestration and the weaving between adaptation checking and actions with the execution operations. Currently, this is done in an ad-hoc way.

- More generally, I would have liked to see a discussions of limitations or drawbacks of your approach. For example, in general in adaptive systems there is a notion of a stable state that needs to be reached before certain adaptation actions (such as replacing a componend) can be performed. I would expect something similar to be the case for your adaptive i-DSMLs, but couldn't see a discussion of this in the paper. More generally, how can one ensure safety of adaptations (the problem that eventually led most of us to discount uncontrolled self-modification of code, despite its appeals)? How does one reason about such models except by executing them?

A12. Ensuring that an adaptation can be processed or leads to a correct state of the model is of course essential. However, as written in the A1 section above (Meta-review), this is a specific point that will require a complete dedicated paper. Nevertheless, as a quick answer, we have already some ideas and developed some techniques related to this issue:

- We have proposed in [4] to use contracts for verifying the adaptation of a model execution.
- Adaptation checking could also be used to make such verifications.
- Some of our prior work addressed this issue in the context of state machines (added reference [1]). The MOCAS platform allows stopping and restarting an executed state machine at runtime for ensuring a consistent adaptation of the state machine.

Review 3

The paper discusses how interpreted DSMLs (i-DSML) can be used in specification and implementation of self-adapting systems. A general scheme of defining adaptable i-DSML is given and illustrated by an example of adaptable state machines.

The topic is interesting and non-trivial. The authors try to give a general characterization of an approach for building adaptable i-DSMLs. I see the current state of the work still immature since both the conceptual foundation and the technical details need further clarification.

The context of the research is self adapting systems where a system that contains a model of itself is dynamically adapted. The general conceptual framework of this type of applications needs a careful clarification in the paper. The authors state that when the model is directly executed (by interpretation) then the model becomes the system. This statement raises confusion and can easily be seen as negating the basic principles of modeling. In general, a model is different from the system and is an abstraction of the system. The abstraction principle postulates that some information about the system is not considered and not represented in the model. Therefore, we cannot put an equality/sameness between the model and the system. If a model is executed then we can obtain some information about the behaviour of the system but still the model and the system are different (the system may even not exist yet in case of prescriptive models).

A13. First, as explained in answer A8 (first reviewer) and in section 2.3 of the revised paper, executable models are particular models. They must be sufficiently enriched (at the lowest level of details) to be substituted to the code. So, we are not discussing in the paper models in general but only executable models and their particular nature.

Second, saying that an executable model is the system is not fully true but is an acceptable shortcut for giving in the introduction the main idea of our approach. To explain this, let us begin with an analogy. A Java program is not directly executed; there is a virtual machine (JVM) that is interpreting it. Here, the running system is precisely the JVM interpreting the Java program. However, everybody says commonly that the Java program is executed and consequently, it is then considered as the running system. This is an acceptable shortcut and this is exactly the same in case of model execution: the running system is the execution engine interpreting a given model but we say that the model is executed and is the running system.

To go further on the relationships between the system and the executed model, this model is actually the abstraction of the system, that is, the abstraction of the compound of itself and of the execution engine interpreting it. Indeed, as the main behavior of a Java program is defined by its code, the main behavior of an executed model is defined by its elements. So, the model, as any model, is an abstraction of a system but, here, in a tricky way: the model represents itself under execution and abstracts away the behavior of the execution engine (i.e., the operational semantics). For this reason, it is an acceptable shortcut to consider the model "as being" the system. We might have this explanation in the introduction but we think that this will lead to lose the reader in subtle and complex details that are irrelevant in an introduction focusing on general ideas.

I suppose that the process of execution supplies the delta that would provide the details dropped in the abstraction process. In other words, the interpreter that provides the semantics of the language is responsible for bringing the delta. This is not clear in the paper.

A14. Actually, as said in answer A8, executable models are so specific they contain all the required details for their execution, excepting the operational semantics implemented by the execution engine. Consequently, there is no "details dropped in the abstraction process". To better answer to this question, section 2.3 of the revised paper try to clarify the very specific nature of executable models.

Furthermore, the authors are not explicit about the causes of adaptations. Generally they may be changes of system environment or changes in system requirements. The concepts of environment and requirements are not explicitly discussed.

A15.We hesitated in introducing the concept of "environment" in our explanations and characterizations (figures 2 and 5). We decided to not add this because it is a minor point of our characterization. Moreover, we do not want to restrict adaptation issues to environment changes as adaptation reasons can be manifold. However, throughout the paper, our examples deal with an adaptation to a changing environment.

Concerning system requirements' evolution, we consider it as a design-based evolution (i.e., functional change); it is then out of the topic of runtime adaptation from our point of view.

The paper is rather abstract concerning the technical details how the execution semantics and adaptation semantics can be implemented. Is there a general approach to this? The paper mentions the availability of an implementation in Kermeta but no details are given.

A16. As written in answers A10 and A11, we plan to define a DSML that will enable the orchestration and the weaving of the adaptation checking and actions with the execution operations. Currently, it is done in an ad-hoc way. Moreover, there was not enough room left to give Kermeta code excerpt. However, as mentioned in the paper, the complete implementation is available online:
http://web.univ-pau.fr/~ecariou/adapt-iDSML/

Other comments:
- talking about adaptation of executed model is strange since it can be interpreted as changes in the final state of the model after the execution is complete. I think the authors mean adaptations on model BEING executed.

A17. Yes, you are right, the exact term is "the model being executed". Talking about "executed model" is a shortcut. Nonetheless, we guess that this shortcut is understandable and does not provide ambiguities. Moreover, it is commonly used in papers discussing model execution.

- the distinction between domain-level and business-level adaptation scope is unclear to me

A18. As written in answer A6, everything about the adaptation scope has been removed from the revised paper.

- Section 3.1 is too abstract and difficult to follow

A19. It is difficult to make these explanations more readable within the same place. Another solution would have been to detail the examples all along the section by illustrating each point of the characterization through the associated examples. But this will lead to have a less concise description as being spread out on more pages.

Review 4

This paper deals with a characterization of adaptable, interpreted domain specific modelling languages (adaptable i-DSML). A conceptual framework for such adaptable i-DSML is defined and illustrated using a simple train example.

A major drawback of the paper is that the approach is discussed at a level that is too informal for getting a clear understanding of the concepts and constructions. For example,
the three items characterizing i-DSML in Section 2 are too vaguely formulated, e.g. "Executing the model makes sense." It is unclear to me how to check whether such an assumption would hold for a given operational model.

A20. These three items are stated in the next three sub-sections (two pages long). This was not perhaps clear, so we have introduced the goal of the sub-sections just after the list of these three items.

The discussion of domain- and business-level scoping is also unclear to me. Therefore, a sentence like "Even if we do not definitively reject the possibility of having business-level scoping in execution semantics, we did not yet succeed to prove its soundness." is very difficult to comprehend. The problem is that the concept business-level scoping is too informally defined and that its soundness property is not defined at all.

A21. As written in answer A6, everything about the adaptation scope has been removed from the paper. It was a secondary point.

Some, more detailed comments.

Page 1, bottom. "correct defaults" ?

A22. It has been replaced by “fix problems”

Page 2. The real different between (a) and (c) in the figure is not clear to me. Elaborate on this difference. By the way, "Figure 1" should be written with capital F.

A23. As written in answer A1 (Meta-review), the introduction has been modified to have better explanations on this point. Actually, the case (c) of Figure 1 is just a restricted version of the case (a) where the system is a model execution system instead of any particular system.

Page 3. The 3 'bullets' in the start of the Section 2 are too vaguely formulated.

A24. See answer A20.

Page 4. Is Fig. 2 a conceptual framework for rather that a definition of an i-DSML?

A25. Yes, you are right, it is a better formulation. We have renamed the figures.

Page 5. Your choice of state machines as an example of an executable DSML modelling language is probably not the best one -- it is a well established model of computation that is not bias towards any particular domain.

A26. Indeed, state machines are well-known. However, “our” state machines can be considered as a DSML. This is mainly because we only use a specific subset of the UML state machines features (composite states, history states…) making these state machines specific, and not equivalent to the UML state machines. Moreover, the word “domain” in the DSML acronym does not necessarily deal with a business domain but with a specialized domain, e.g., a particular subset of UML state machines.

Page 8, 1. par. "their nameS are"

Page 8, mid. In which sense is an adaptable i-DSML a logical extension of an i-DSML?

A27. In the sense of the next sentence in the paper: "Indeed, adaptable models are executable models endowed with adaptation capabilities." An adaptable i-DSML is firstly an i-DSML that contains additional elements for the adaptation. From this point of view, an adaptable i-DSML extends an i-DSML.

Page 8, bot. "modelizes process" check sentence.
Page 9, point 2). It is not clear to be what this really means.

A28. This has been corrected for the first point. We agree that the second point is rather abstract when introducing section 3 but it is fully described all along the rest of the section.

Page 9, bot par. The sentence "We consider only substitutions ... not feasible to define new semantics from scratch ... ". As I understand your approach the i-DSMLs are treated as data and you have full flexibility. If I'm right about this it seems feasible to defined a new semantics.

A29. The executed model can be seen only as data. This is why it can be modified without restriction (addition, removing or modification of elements). However, the execution semantics is implemented under the form of lines of code (in Java, Kermeta…) within the engine. As written in the paper, defining a new semantics consists in defining new lines of "hand-written" code. We do not see how it can easily be done in an automatically manner. For this reason, we think that it is only possible to use semantics variants out of a set of already defined variants already implemented in the engine.

Page 11, mid. "Even if ... prove its soundness". I don't understand what it means. How is the soundness formulated?

A30. It is just a speaking formula. There is no formal proof or similar things here. Anyway, this part has been removed from the paper as done for "dealing with adaptation scope" (see answer A6, first reviewer).

Page 14, adaption actions, train example. It seems to me that it works because your train model have the simple property that transitions on a specific event are all ending in the same state.

A31. Yes, you are right. This is what we explained in the footnote 5. This is not a problem to make this assumption because it can be considered as defining a special kind of state machines. Following the DSML spirit, you build the modeling language that is exactly fitting your requirements.