Department of Software Engineering
Faculty of Information Technology
VNU University of Engineering and Technology
144 Xuan Thuy, Cau Giay, Hanoi, VIETNAM

March 10, 2023

Prof. Günther Ruhe
Editor Head of INFSOF

Dear Prof. Günther Ruhe:

We thank the reviewers for their interest in our work and for their helpful comments that will improve the manuscript. We have revised the paper in accordance with the reviewers suggestions. Those revisions are described below.

Thank you.
Sincerely,
Duc-Hanh Dang

Reviewers’ Comments

With the Author's Revision Notes

General comments

Their general opinion is that after a revision addressing the reviewer's comments and after performing another round of peer evaluation, the manuscript could form a valuable contribution to the Journal.

1 Reviewer 3

1.1 Summary

The authors propose a domain modeling method approach that integrates the structural and the behavioral aspects of a domain, to generate Java code. This approach is tooled by a open source prototype, named jDomainApp.

The paper is organized in 11 sections that should be reorganized to improve readability. Moreover, the text contains several typos and small inaccuracies that makes it unnecessarily hard to read.

The paper has several issues that require deep changes:
- The main contribution is not clear, leading to other issues listed below;
- The motivation is not convincing;
- The related work does not compare the approach with similar ones, it is too generic;
- The evaluation is too shallow and is not convincing;
- The approach is tooled with a open source prototype, but the paper does not highlights the implementation.

1.2 Section 1 - Introduction

The first section (Introduction) explains the motivation and lists the paper's main contributions. Here, the authors should clarify their main contribution: the title names AGL (Activity Graph Language), text says the contribution is a method (a novel unified domain modeling method) and the end of the introduction lists several contributions.

Yes, we updated the text in order to emphasize the main contribution at two main points: (1) a mechanism to incorporate behavior aspects for a unified domain model, in which a new aDSL named AGL is defined in order to capture the domain behaviors; and (2) a unified modeling method for domain-driven software development.

The authors should also clarify their motivation. The fact that no other DDD approach integrates structural and behavioral aspects is a good thing, but it's not a good motivation. They should explain why this integration is needed.

Yes, we agree with the reviewer that it should be a good thing that no other DDD approach for a combination of structural and behavioral aspects (in the sense that each model should be separately captured from one certain and specific perspective. And then, transformations helps to relate them with each other). We emphasize the main benefits of this integration at two points: (1) we could obtain a composition of domain concerns for an executable version of the software at higher level of abstraction, which in turn significantly eases software construction from the domain model. (2) it eases the definition of transformations from a domain model incorporated with behavior models (in UML and DSLs) to construct the software.

The presentation of AGL is very confusing. Consider the following sentence: "We then propose a novel horizontal aDSL, named activity graph language (AGL) to express the UML activity graph."

I suppose the authors refer to the UML "Activity diagram", but sine it already has a notation, why AGL is needed? In the rest of the paper, the authors use several times "activity graph", and the reader is unable to find if they refer to the UML Activity Diagram or to the AGL

Yes, we have re-expressed this point: "we define a novel aDSL, named AGL (Activity Graph Language), to represent behavioral aspects (that could be captured using UML activity diagrams and statecharts) and to incorporate them as part of the unified domain model."

1.3 Section 2 - Motivating Example and Background

Section 2 presents a running example and some background concepts. This section should be reorganized and improved. In on hand, the idea of having this running example is interesting and improves the readability of the paper. In the other hand, it should present the **real technological background, domain-driven design, and it should not present aDSL,** since it is a contribution of the current paper.

Yes, we have re-organized this section as the reviewer's suggestions. We also added a new part to provide a background about DDD. We improved the explanation of the motivating example in order to better highlight the motivation for this work.

The class diagram presented in Figure 1 has some issues. First, I suppose that it is a UML class diagram, is it correct? If this is the case:
- the type "int" should be replaced by "Integer"
- the type "AuthorizStatus" does not exist
- Why do you use notes instead of more appropriate UML concepts, such as Stereotypes, Tags and OCL to specify mutability, value ranges, etc. ?
Moreover, in Figure 2:
- the initial and the final nodes are missing

Yes, we have updated these figures for these points. (Note that AuthorizStatus is an enumeration type that is employed in the class Authorisation.)

Something that is confusing here is that in UML, Activities are linked to a UML "Behaviored Classifier", thus the **modeler doesn't need to combine them to create a "Unified Domain Model", since he is already creating such model**.

Yes, we have updated the text to make clear this point: "Since this domain behavior is currently captured in UML, we would need a further mechanism to maintain a consistency between the two models, toward composing them, normally at an implementation level. As an alternative approach for this aim, following the DDD approach introduced in our previous work [4], we would consider such a behavior concern as an extension of the essential domain model for a unified domain model (i.e., a DDD with the key features, feasibility, productivity, and understandability)."

Section 2.2 presents DCSL, the domain class specification language.
I understand that this is not a contribution of the paper, but what are the **differences between DCSL and UML**? Why to define a new modeling language that seems to be very similar to UML?
-> You should **clarify the differences between your modeling languages and UML**.

Yes, we have updated Sect.2 to make clear this point. Specifically, SubSect.2.1 presents the requirements to realize DDD (raised by Evans). SubSects.2.2-3 explains our approach for DDD: DCSL is proposed to specify domain models in the context of MOSA. SubSect.2.4 highlights the motivation for this work.

Section 2.3 presents MODA, the Module-Based Software Architecture, which basically transforms domain classes into "models" of the MVC framework, to create a running software.

In addition to that, the MOSA would help us to achieve the domain model as a DDD with the main features, feasibility, productivity, and understandability.

Section 2.4 lists two challenges the should be tackled to realize the proposed approach.
In this section, "DCSL" is no longer referred to as a "language" but as a "framework".
-> I suggest to move some parts of the current related work, such as MVC and Attribute-Oriented Programming to the background section.

Yes, we have updated the text (SubSect.2.4) to make the point more clear. We also refer to DSCL as a domain definition language.

1.4 Section 3 - Overview of the Proposed Approach

Section 3 presents the proposed approach, providing 2 definitions: "unified class model" and "unified model"

Yes, this section overviews our approach and provides the definition for "unified class model" and "unified model". We have also updated this section to make it more clear the basic idea of our approach with such the main questions: Within the proposed mechanism which kind of domain behaviors is incorporated into the domain model? How do we capture their semantics domain in the context of MOSA?

1.5 Section 4 - Module Action Semantics

Section 4 presents the "Module Action Semantics", an action language **based on the UML Action Language**. It defines "Atomic Action" and introduces 8 core atomic actions, which are combined later to form an Atomic Action Sequence.
Again, it's hard to understand the differences between "Module Actions" and "UML Actions".
- Are they the same?
- Is it possible to use the UML Actions to specify the core module actions?

Yes, in this updated version (SubSect. 3.2.) we clarify the semantics domain of domain behaviors, that is more specific and narrower than UML Activity diagram's semantics. Specifically, we would customize and restrict UML Activity diagram (using the DSL language AGL) in order to make them fit well for capturing the behavior semantics of the activity module (as a composite module in MOSA to coordinate a collaboration among modules w.r.t the activity model).

1.6 Section 5 - Domain Behavior Patterns

Section 5 introduces "Domain Behavior Patterns", which seem to correspond to the UML Control Nodes.
The authors chose to only present one pattern, Sequential and leave the others to the "long version" of the paper.
- What do you mean by "long version"? Did you submit it to another journal?
- If you want to present only one, **why don't you choose something more complex, such as the Forked and Merged patterns**?

Yes, we choose the pattern Decisional Pattern that is complex enough to illustrate the basic idea of our method. For a detailed explanation of the four remaining patterns, we would refer the reader to the technical report of this paper.

The use of UML templates to configure the unified model seems to be an interesting choice.
I strongly suggest the authors **to improve the explanation of this mechanism**.

Yes, by focusing on a more complex pattern like Decisional Pattern we would improve the explanation of the mechanism to be more understandable. In line to the aim, we supplement a procedure to transform the input activity diagram to an AGL specification.

1.7 Section 6 - Module-Based Domain Behavior Language

Section 6 presents AGL (Activity Graph Language), the Module-Based Domain Behavior Language. The presentation is done in terms of the abstract and the concrete syntaxes, as well as its semantics. This Section also presents the Java and C# annotations used to translate AGL to these languages. This section clarifies the Section 4 and **it's hard to understand why the authors chose to present AGL in two different sections**.

Yes, basically we could put the AGL metamodel in Sect. 4. However, the metamodel should characterize the domain semantics of AGL to be narrower since it is restricted by domain behavior patterns, that explained in Sect. 5. That is why we present AGL both syntax and semantics in Sect.6.

The Abstract Syntax is presented as an extension of the UML Action Language. Figure 10 presents the AGL meta-model in an unknown modeling language, which seems merge Java and UML concepts.
-> I strongly suggest the **authors to use either UML, MOF, or Ecore** to specify this meta-model

Yes, we have updated the AGL metamodel and specified using Ecore. Note that the current version of the metamodel is used mainly for documentation purpose. We plan to obtain AGL specifications by transforming from UML Activity diagram in future work. At that time, the AGL metamodel should be precise enough for the goal.

Section 6.2 presents the concrete syntax (CSM, Concrete Syntax Model) in terms of a set of annotations. The term "concrete syntax" is a bad choice, since this section actually presents an abstract syntax, the one from the annotation used in the generated code. Again, the syntax is presented as a class model in an unknown modeling language (Figure 12). The text says the Figure is a UML model, although it doesn't respect its syntax.
-> I strongly suggest the authors **to use either UML, MOF, or Ecore** to specify this meta-model

Yes, we have updated the metamodels using Ecore. Due to the limited space, a detailed explanation of the metamodels is only shown in the technical report for this paper.

Section 6.3 presents the textual concrete syntax of the Annotations.
- What is the **purpose of this section**, since the concrete syntax of Java (and C#) annotations are already known?

Yes, the concrete syntax of AGL is based on Java Annotation. This section aims to make it clear which annotations are employed for the AGL specification and the relationships between them.

Finally, Section 6.4 quickly presents the semantics of AGL as an extension of the UML one.
This section also contains a **definition of the Software** Generated in MOSA.
- What is the **relation between this definition and the AGL semantics** ?

We have removed this section since in the updated version of this paper the content of AGL's semantics has been more clearly explained in Sects. 4 and 5.

1.8 Section 7 - Tool Support

Section 7 briefly presents the JDA framework, the tool that supports the proposed approach. **This section is quite disappointing**, as it only presents the general structure of the framework. It would be more interesting if it rather presents **design choices and used technologies**. For instance, this section could answer some of the following questions:
- How annotations are inserted into Java or C# code?
- How the AGL is implemented?
- What is the input format?

Yes, thank you very much for the comments. We have carefully updated this section. Moreover, we add the case study OrderMan in order to illustrate the tool support as well as to evaluate our method in Sect. 8.

1.9 Section 8 & 9 - Evaluation & Threats to validity

Section 8 presents the Evaluation of AGL, in terms of expressiveness, required coding level, and constructibility. Section 9 discusses the Threats to validity. **The evaluation is not convincing,** I strongly suggest the authors **to perform experiences to evaluate AGL and compare it to similar languages.**

Yes, thank you very much for the comments. We have carefully revised this section in order to make it be convincing.

1.10 Section 10 - Related Work

Section 10 presents the related work, organized in several topics: DSL engineering, DDD, Behavior modeling with UML, UML, MDSE, Attribute-oriented programming, etc.
This section is very **generic and does not compare AGL with similar languages/approaches**. For instance, *it's hard to tell in what AGL and MVC are related*.

Yes, we have moved some part of this section (including MVC) to the background section.

1.11 Minor remarks

=== General
- The text has too many acronyms, some parts are hard to parse

Yes, we have removed several acronyms for easy reading.

=== Introduction
- Why "aDSL" and not "ADSL"?
- "and consists in a set of annotations" -> "and consists of a set of annotations"
- "The resulted domain model" -> "The resulting domain model"
- "UML activity diagrams and statecharts" -> "UML Activity and State Machine diagrams"
- "We view unified model" -> "We view the unified model"
- "we introduced UML activity diagram" - > "we introduced UML activity diagrams"
- "we choose UML activity diagram" -> "we choose the UML activity diagram"
- "because this language has been shown to be domain-expert-friendly and that it can be used to design " -> "because this language is domain-expert-friendly and is used to design"
- "A key benefit from combining" -> A key benefit of combining
- "that includes a very few number of concepts." -> "that includes few concepts."
- "Section 2 presents our motivating example and a technical background." -> "Section 2 presents our motivating example and the technical background."
- "Section 4 provides a formal semantics for module actions" -> "Section 4 provides formal semantics for module actions"

Yes, we have updated the text as the reviewer's suggestions. We still use the term aDSL to keep consistency with our previous paper (part of the aim is to highlight the keyword DSL).

=== Section 2
- "motivates our work by means of example" -> "motivates our work through an example"
- "We will introduce here" -> "We introduce here"
- "specialised" -> "specialized" (I suppose you are using American English spelling)
- "labelled" -> "labeled"
- "Figure 2 illustrates the combination of a simplied class diagram" -> "Simple"? "Simplified"?
- "Finally, meta-concept Domain Method is composed of Method with a certain behavior type. " -> I don't understand this sentence: Finally, meta-concept Domain Method is composed of Method with with a commonly-used constraints and behaviour types that are often imposed on instances of these meta-concepts in a domain model
- "Domain Field that realises" -> "Domain Field that realizes"
- "nonannotation" -> ?
- "initialisation" -> "initialization"
- "Both of them are assigned with a DClass element, which state" -> "Both of them are assigned with a DClass element, which states"
- "which references the name of domain field" -> "which references the name of the domain field"
- "The existing DDD frameworks [2, 3] supports" -> "... support"
- "must be backbone" -> "must be the backbone"
- "parameterised classes" -> "parameterized classes"
- "to automatically generate a software." -> "to automatically generate software."
- "According to Booch [18], an object-oriented software consists in" -> "According to Booch [18], object-oriented software consists of"
- "the rest of MOSA model via activity graph" -> " the rest of MOSA model via an activity graph"

Yes, we have updated the text as the reviewer's suggestions. We also rewrite the sentence "... is composed of Method and commonly-used constraints and behaviour types". Moreover, when explaining the example with Fig. 2, we also give an example for behavior type: "Each method is assigned with a DOpt element, which specifies the behavior type. For instance, method genId, whose behavior type is AutoAttributeValueGen, ..."

=== Section 3
- "The activity graphs are expressed in the language AGL that will be explained in Section 6." -> "The activity graphs are expressed in the language AGL, which is explained in Section 6."
- "feedbacks" -> "feedback"
- check the usage of "w.r.t.": you seem to use as an abbreviation of "respectively"
- "the resulted unified model" -> "the resulting unified model"
- "the two actions nodes" -> "the two action nodes"
- "In other word" -> "In other words"

Yes, we have updated the text as the reviewer's suggestions.

=== Section 4
- For citing a specific page, use something like: "\cite[p.~441]{uml:omg:2017}.". No need to specify the chapter or section.
- "State is argued to be an intrinsic part of behavioral specification" -> "A State is an intrinsic part of the behavioral specification"

- "Note the followings" -> "Note the following"
- "For exposition purpose," -> "For presentation purposes,"
- "The second group include" -> "The second group includes"
- " which is more formally modelled" -> " which is more formally modeled"
- "The definition of ASE gives rise to a notion of reachable state," -> "The definition of ASE gives rise to the notion of reachable state,"
- "actions cannot reach state Opened" -> "actions cannot reach the state Opened"
- " This is because this action concerns only with inputting data" -> "This is because this action concerns only input data "
- "has only one reachable state, which are their own states" -> "has only one reachable state, which is their own state"
- " actions that precedes" -> " actions that precede"
- "that starts with action newObject" -> "that starts with the action newObject"
- " is more formally modelled" -> " is more formally modeled"
- "The main reason is that Activity diagram" -> "The main reason is that the Activity diagram"

Yes, we have updated the text as the reviewer's suggestions.

=== Section 5
- "behavior specification in UML activity diagram" -> "behavior specification in the UML activity diagram"
- "we only focus on Sequential Pattern" -> "we only focus on the Sequential Pattern"
- "nonannotation" -> ?
- "for user to enter input" -> "for the user to enter input"
- "allowing user to re-enter" -> "allowing the user to re-enter" (twice)

Yes, we have updated the text as the reviewer's suggestions.

=== Section 6
- "From the language engineering's perspective" -> "From the language engineering perspective"
- "to incorporate activity graph into MOSA" -> "to incorporate the activity graph into MOSA"
- "conditions on decision node" -> "conditions on decision nodes"
- "variable is alternative to using object flow" -> "a variable is an alternative to using object flow"
- "sub-set" -> "subset"
- "that realizes the the association" -> "that realizes the association"
- "This class is used to specify behavior" -> "This class is used to specify the behavior"
- "Actual implementations of the interface Decision w.r.t Join is provided" -> "Actual implementations of the interface Decision w.r.t Join are provided"
- "labelled" -> "labeled"
- "From the practical standpoint" -> "From a practical standpoint"
- "the language user to construct" -> "the language used (?) to construct"
- "the default state value of action" -> "the default state value of the action"
- "For many cases," -> "In many cases,"
- "including action and the control types." -> "including action and control types."
- "grey-coloured" -> "grey-colored"
- "the domain logics" -> "the domain logic"

- "the combination of unified model and activity graph" -> "the combination of a unified model and an activity graph"

Yes, we have updated the text as the reviewer's suggestions.

=== Section 7
- "First, model manager" -> "First, the model manager"
- "Second, view manager" -> "Second, the view manager"
- "organising" -> "organizing"
- "Third, object manager" -> "Third, the object manager"
- "to/from an external storage" -> "to/from external storage"

Yes, we have updated the text as the reviewer's suggestions.

=== Section 8
- "Thus, we will measure" -> "Thus, we measure"
- "one third" -> "one-third"

Yes, we have updated the text as the reviewer's suggestions.

=== Section 9
- "The domain experts give feedbacks" -> "The domain experts give feedback"
- "Usability of the software GUI" -> "The usability of the software GUI"
- "view-point" -> "viewpoint"
- "favour" -> "favor"
- "Evolution of languages" -> "The evolution of languages"
- "Selection of the unified" -> "The selection of the unified"

Yes, we have updated the text as the reviewer's suggestions.

=== Section 10
- "We position our work at the intersections" -> "We position our work at the intersection"
- "can be classified based on domain" -> "can be classified based on the domain"
- "such as sequence diagram" -> "such as sequence diagrams"
- "Evans only uses sequence diagram" -> "Evans only uses sequence diagrams"
- "However, ApacheIsIs" -> "However, ApacheIsis"
- "modelled" -> "modeled"
- "looks at the similar view" -> "looks at a similar view"
- "behavioural" -> "behavioral"
- "Software that are" -> "Software that is"
- "in which a software is composed from a hierarchy" -> "in which software is composed of a hierarchy"

Yes, we have updated the text as the reviewer's suggestions.

== Section 11
- "to incorporates" -> "to incorporate"
- "the domain-specific features of UML activity diagram" -> "the domain-specific features of the UML activity diagram"

- "in a previous work" -> "in previous work"
- "to construct for the model an UML activity graph" -> ? to express the domain behaviors for a unified model
- "DDD in two important fronts" -> "DDD on two important fronts"
- "horizontal DSLs which can be used" -> "horizontal DSLs that can be used"

We position our work at

Yes, we have updated the text as the reviewer's suggestions.

2 Reviewer 4

2.1 Summary

In their previous work, the authors developed an internal DSL (called DCSL) to define domain models and generate code from them. In this paper, the authors complement this language with another internal DSL (called AGL) that permits attaching behaviour to domain classes. Besides the design of the new DSL, the paper briefly reports on its implementation and evaluation.

PROS: The extension with respect to the authors' previous work is substantial. The proposed DSLs are useful and well designed. The manuscript is generally well written. Although some parts are a bit difficult to follow, the paper includes examples to illustrate the core ideas.

CONS: The paper does not convincingly justify the advantages of the proposal compared to other non-annotation-based approaches (e.g., model-based ones). The **evaluation is very weak**. The paper illustrates the approach using a **minimal running example**, but it does **not validate the expressiveness or usefulness** of AGL on **realistic case studies**.

2.2 Section 1 - Introduction

Section 1. **The paper does not convincingly justify the advantages** that the proposed approach has in comparison to the state-of-the-art in general (i.e., looking beyond annotation-based languages). In the MDSD/MDE area, there are many proposals for defining different aspects of an application (domain concepts, behaviour, GUI...) and generating code. The only apparent difference is that DCSL and AGL are internal DSLs. The introduction should justify better the novelty w.r.t. existing generative approaches not based on annotation-based DSLs, and the benefits of using internal DSLs.

Yes, we have updated the introduction to better highlight the main contributions of this work. We emphasize that (1) "current model-driven approaches can be employed to compose a domain model with other concerns expressed at either a high level using a general language like UML and DSLs. The final program then could be obtained by model transformations, either model-to-model or model-to-text ones. This work focuses on an alternative approach for this aim as a refinement of an aDSL-based software development method for DDD that we proposed in a recent work." and (2) "We aim to define an extension of domain model that allows us to represent behavior aspects of the domain within a unified model. The benefit of this extension is that we could obtain a composition of domain concerns for an executable version of the software at higher level of abstraction, which in turn significantly eases software construction from the domain model."

2.3 Section 2 - Motivating Example and Background

Section 2.4. The stated challenges are DCSL-specific. Instead, they could be reformulated as general challenges in the area: how to extend a domain definition language to add behaviour, and how to incorporate this behaviour in domain models.

Yes, we agree with the reviewer on this point (indeed a great idea) and have updated the text for it.

2.4 Section 3 - Overview of the Proposed Approach

Figure 6. For readability, the diagram to the right could show the role of each class (data store, activity class, decision class) as class stereotypes.

Yes, we have updated Figure 6 as the reviewer's suggestions.

2.5 Section 5 - Domain Behavior Patterns

Section 5. Clarify how patterns are to be used: either as a guide (a cheat-sheet) for AGL users, or as code templates for the automatic generation of AGCs starting from a higher-level definition (such as the diagram on the left of Figure 8).

Yes, we have supplemented a procedure to transform the input activity diagram (as a higher-level definition of domain behaviors) to an AGL specification (an AGC).

2.6 Section 6 - Module-Based Domain Behavior Language

Figure 10. The name of abstract classes should be in italics.

Yes, we have updated the metamodels (models) using Ecore (UML). The remark helps us to fix some places, e.g., for the abstract class "ControlNode".

Section 6.1. **Where is the actual implementation of the methods of the Decision/Join interfaces done,** at the specification level or at the code level? More generally, do AGL users have to complement the AGL specifications with handwritten code?

Yes, we have updated the AGL metamodel and specified using Ecore. Note that the current version of the metamodel is used mainly for documentation purpose. We plan to obtain AGL specifications by transforming from UML Activity diagram in future work. At that time, the AGL metamodel should be precise enough for the goal.

2.7 Section 7 - Tool Support

Section 7. **It is difficult to assess the usability of a DSL without seeing a complete example of the DSL usage.** In this respect, I highly recommend including the code of the running example in an appendix or a website. At a first glance, the code fragments shown in the previous sections seem complex to define. The paper could discuss briefly the usability of the language. In addition, I miss a screenshot of the development IDE.

Yes, we have revised this section based on your suggestions. The comments indeed are very helpful and valuable. Thank you very much!

2.8 Section 8 - Evaluation

Section 8. **The evaluation is very weak.** Firstly, there are **no research questions**. Second, the **expressiveness of AGL is argued but not evaluated**. To demonstrate expressiveness, ***the paper could***

report on a couple of realistic case studies showing the extent to which AGL can capture all the necessary behaviour without resorting to code. For example, the approach seems limited to CRUD applications, so it is not clear whether **more complex behaviour** could be achieved without coding. Third, **compactness** is evaluated by comparing the concrete and abstract syntax representations of the DSL. This is fine, although it would have been more convincing to **compare with other approaches** (not necessarily based on aDSLs). Finally, **Section 8.3 should be moved out of the evaluation since it is more of a discussion than an evaluation**.

Yes, we have realized that this is a quite big limitation of this text. Thanks to the suggestions, we have made enhancements in order to get it over this point.

2.9 Section 9 - Threats to validity

Section 9.1 would fit better in Section 8, as a last subsection named "Discussion and Limitations". In addition, I would suggest adding one paragraph discussing **the generality of AGL**.
What would be the effort to implement the approach in other host languages?
What features must a host language have to allow implementing DSCL/AGL on top of it?

Yes, Sect. 8 has been updated based on the reviewer's suggestions.

Section 9.2 could be structured into validity types **(internal, external, construct).**

Yes, we have revised this section as the reviewer's suggestion.

2.10 Section 10 - Related Work

Section 10. The bibliography is relevant but somewhat old. Including more recent works from the last 5-6 years (other than by the authors) would demonstrate that the topic is timely. In addition, the paper could mention the possibility to use language frameworks (e.g., Xtext, GEMOC, Monticore...) to define external DSLs, as an alternative to the use of annotation-based DSLs.

Yes, we have revised this section as the reviewer's suggestion.

2.11 Other remarks

p1: resulted domain -> resulting domain
p2: and that it can be used -> and it can be used
p2: by means of example -> by means of an example figure 1: regsiters-to -> registers-to
p6: frameworks [2, 3] supports -> frameworks [2, 3] support
p10: actions nodes -> action nodes; p11: in in -> in
p11: f is name -> f is the name
p11: The second group include -> The second group includes
p18: the the -> the
p21: whose the activity -> whose activity
p26: helps fills -> helps fill

Yes, we have updated the text as the reviewer's suggestions. Thank you very much!