Activity) can be specified. The embedded editor only shows the edited statements, while directly dealing with the serialization in the Alf unit corresponding to the edited Activity.

Limitations: This solution is a kind of workaround. A more permanent approach might be to refactor the Xtext implementation of the grammar into something more modular. Further investigations are however required to determine the feasibility of this modular approach (realizations made in the Xtext community around Xbase[13] suggest that it would be feasible), as well as its development cost.

C. Combining with profiles

Problem statement: Profiles can be used to implement DSLs on a UML basis. As shown in the examples of section II.A, these UML-based DSLs might be executable, so that the question of the use of Alf in the context of profiled UML models naturally arises.

Key Alf aspects: This use of Alf is considered in the OMG specification. The support in terms of stereotype application is however relatively limited. Alf stereotypes can only have attributes that are typed with primitive types or one attribute typed by a meta-class. In addition, stereotype attributes that are typed by a primitive type can only have a single value.

Proposed solutions: The proposed solution consists in extending the syntax, validation and compiling rules of Alf, consistently with the abstract syntax and the semantics of the considered UML profile.

Experiments: In [5] we have studied a refactoring of the Value Specification Language (VSL) as an extension of Alf. VSL has been standardized in the context of the UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE)[14]. The main rationale for VSL was to provide users from the real-time domain with a simple textual syntax for specifying the values of non-functional properties of their system models. In VSL, rules for producing typed expressions mainly address aspects that are not specific to the real-time domain. They are related to general-purpose concerns that are also considered by Alf. Our proposal consisted in properly defining VSL as an extension of Alf, focusing on the aspects of VSL that are valuable for the real-time domain, and thereby leveraging the Alf specification and implementation. In a proof-of-concept implementation, we demonstrated how Alf syntax and validation rules could be extended for that purpose. The experiments were limited to the Expression subset of Alf.

Limitations: While the feasibility of the approach has been demonstrated, its applicability is limited by our current Xtext implementation of the Alf language and the fact that it is monolithic (as discussed in section III.B). The parts that would need to be extended or specialized for a particular domain cannot be easily extracted. More generally, Alf is a complex language, so that any extension (from both syntactic and semantic standpoints) should be considered with great care and may require a lot of development effort. It is not clear today if such extensions of Alf should be encouraged, and if evolution in the specification or implementation could make such extensions easier and safer.

[13] https://wiki.eclipse.org/Xbase
[14] http://www.omg.org/spec/MARTE/

IV. Related Work

We have built a modeling and simulation tool based on the OMG standards fUML, PSCS and Alf. During the definition of this tool, we identified challenges (extensibility, control, time support and connectivity with external tools) related to the use of the fUML standard for simulation, and we proposed solutions for them. In the literature, we have selected two tools that, by construction, should address challenges similar or very close to those we identified: Moliz[15] and Gemoc Studio[16]. These tools have been selected either for their strong link with fUML (Moliz) or their Eclipse-based implementation (both tools). It would have been interesting to consider the Cameo Simulation Toolkit[17] (fUML based) as well. However, since this is a commercial tool, we could not have access to the implementation and make relevant comparisons. We discuss in the following sections how these tools address the challenges we identified.

A. Extensibility

Domain specific languages are expressed with UML using profiles. To capture the semantics of these languages, we proposed a systematic approach to extend the fUML semantic model with new visitors. Moliz and Gemoc do not consider profiles. However they both address the definition of the semantics of MOF-based domain specific languages. Moliz uses fUML to define the semantics [7] while Gemoc uses Kermeta [4].

B. Control and observability

During simulation, we needed to be able to observe the execution to make it easier to understand a model. To do so, we proposed delegating, at specific points, the control of the execution flow to a particular entity (e.g. a debugger) to enable the control of the execution.

Moliz and Gemoc propose a similar approach. They both have a connection with a debugging environment based on the control-delegation principle. Furthermore, both tools use this approach to connect with trace generators (specific to fUML in the case of Moliz [6] and specific to each DSML in the case of Gemoc [4]). These two tools share an implementation approach based on aspect-oriented programming, which could be interesting for addressing the limitation we have identified (i.e., identification of control delegation point, as discussed in section II.B).

C. Time support

Time is a key aspect in simulation that is unfortunately not directly supported in fUML. To introduce time support, we relied on the control-delegation approach, with a control entity responsible for the evolution of time and the scheduling of the execution.

[15] http://www.modelexecution.org/
[16] http://gemoc.org/studio/
[17] http://www.nomagic.com/products/cameo-simulation-toolkit.html