A Tool-Supported Approach for Concurrent Execution of Heterogeneous Models

Benoit Combemale[1], Cédric Brun[2], Joël Champeau[3], Xavier Crégut[4], Julien Deantoni[5], and Jérome Le Noir[6]

[1] Inria and Univ. Rennes 1, France
benoit.combemale@inria.fr
[2] Obeo, France
cedric.brun@obeo.fr
[3] ENSTA Bretagne, France
joel.champeau@ensta-bretagne.fr
[4] INPT ENSEEIHT, IRIT, France
xavier.cregut@enseeiht.fr
[5] University of Nice Sophia Antipolis, I3S/INRIA AOSTE, France
julien.deantoni@polytech.unice.fr
[6] Thales Research & Technology, France
jerome.lenoir@thalesgroup.com

1 Context and Approach

In the software and systems modeling community, research on domain-specific modeling languages (DSMLs) is focused on providing technologies for developing languages and tools that allow domain experts to develop system solutions efficiently. Unfortunately, the current lack of support for explicitly relating concepts expressed in different DSMLs makes it very difficult for software and system engineers to reason about information spread across models describing different system aspects [4].

As a particular challenge, we investigate in this paper relationships between, possibly heterogeneous, behavioral models to support their concurrent execution. This is achieved by following a modular executable metamodeling approach for behavioral semantics understanding, reuse, variability and composability [5]. This approach supports an explicit model of concurrency (MoCC) [6] and domain-specific actions (DSA) [10] with a well-defined protocol between them (incl., mapping, feedback and callback) reified through explicit domain-specific events (DSE) [12]. The protocol is then used to infer a relevant behavioral language interface for specifying coordination patterns to be applied on conforming executable models [17].

All the tooling of the approach is gathered in the GEMOC studio, and outlined in the next section. Currently, the approach is experienced on a systems engineering language provided by Thales, named Capella[7]. The goal and current state of the case study are exposed in this paper.

[7] Cf. https://www.polarsys.org/capella/

2 The GEMOC Studio

The GEMOC Studio is an eclipse package that contains components supporting the GEMOC methodology for building and composing executable Domain-Specific Modeling Languages (DSMLs). It includes two workbenches: the *GEMOC Language Workbench* and the *GEMOC Modeling Workbench*. The language workbench is intended to be used by language designers (aka domain experts), it allows to build and compose new executable DSMLs. The Modeling Workbench is intended to be used by domain designers, it allows to create and execute heterogeneous models conforming to executable DSMLs.

The GEMOC Studio results in various integrated tools that belong into either the language workbench or the modeling workbench. The language workbench put together the following tools seamlessly integrated to the Eclipse Modeling Framework (EMF: `https://eclipse.org/modeling/emf`):

- *Melange* (`http://melange-lang.org`), a tool-supported meta-language to modularly define executable modeling languages with execution functions and data, and to extend (EMF-based) existing modeling languages [10].
- *MoCCML*, a tool-supported meta-language dedicated to the specification of a Model of Concurrency and Communication (MoCC) and its mapping to a specific abstract syntax of a modeling language [6].
- *GEL*, a tool-supported meta-language dedicated to the specification of the protocol between the execution functions and the MoCC to support feedback of the runtime data and to support the callback of other expected execution functions [12].
- *BCOoL* (`http://timesquare.inria.fr/BCOoL`), a tool-supported meta-language dedicated to the specification of language coordination patterns, to automatically coordinates the execution of, possibly heterogeneous, models [17].
- *Sirius Animator*, an extension to the model graphical syntax designer Sirius (`http://www.eclipse.org/sirius`) to create graphical animators for executable modeling languages[8].

The different concerns of an executable modeling language as defined with the tools of the language workbench are automatically deployed into the modeling workbench that provides the following tools:

- *A Java-based execution engine* (parameterized with the specification of the execution functions), possibly coupled with TimeSquare (`http://timesquare.inria.fr`) [9] (parameterized with the MoCC), to support the concurrent execution and analysis of any conforming models.
- *A model animator* parameterized by the graphical representation defined with Sirius Animator to animate executable models.
- *A generic trace manager*, which allows system designers to visualize, save, replay, and investigate different execution traces of their models.

[8] For more details on Sirius Animator, we refer the reader to `http://siriuslab.github.io/talks/BreatheLifeInYourDesigner/slides`

– *A generic event manager*, which provides a user interface for injecting external stimuli in the form of events during the simulation (e.g., to simulate the environment).
– *An heterogeneous coordination engine* (parametrized with the specification of the coordination in BCOoL), which provides runtime support to simulate heterogeneous executable models.

The GEMOC studio is open-source and domain-independent. The studio is available at `http://gemoc.org/studio`

3 Industrial Case Study: xCapella

Arcadia (`https://www.polarsys.org/capella/arcadia.html`) is a model-based engineering method for systems, hardware and software architectural design. It has been developed by Thales between 2005 and 2010 through an iterative process involving operational architects from all the Thales business domains. Arcadia promotes a viewpoint-driven approach (as described in ISO/IEC 42010 Systems and Software Engineering - Architecture Description [1]) and emphasizes a clear distinction between need and solution. The *Capella* modeling workbench is an Eclipse application implementing the ARCADIA method providing both a DSML and a toolset which is dedicated to guidance, productivity and quality. The Capella DSML aggregates a set of 20 metamodels and about 400 meta-classes involved in the five engineering phases (*aka.* Architecture level) defined by ARCADIA. The *Capella* modeling workbench is based on Sirius in order to define the graphical concrete syntax of the Capella DSML. *Capella Studio* provides a full-integrated development environment, which aims at assisting the development of extensions for Capella modeling workbench. This studio is based on Kitalpha incubated at Thales for several years before being recently released in open source as one of the PolarSys projects. Kitalpha allows viewpoint designers to extend the Capella DSML. Despite the existence of behavioral models, the Capella modeling workbench does not provide any simulation capability. The Capella behavioral models are limited to: modes and states, functional chain data flows, and scenarios. Neither the behavioral semantics or the coordination between these languages are defined.

3.1 Objectives and overcoming initial limitations

In order to support the execution of models, a dedicated executable concurrent semantics is required. We started with two of the three behavioral languages from the Capella DSML (*i.e.*, data flows and mode automata).

In addition, to capture the interaction between the models conforming these behavioral languages, we specified the behavioral coordination patterns between them (Fig. 1).

Our study proposes to use the GEMOC studio to support system engineers so they can tame system modeling activity and improve the confidence in the

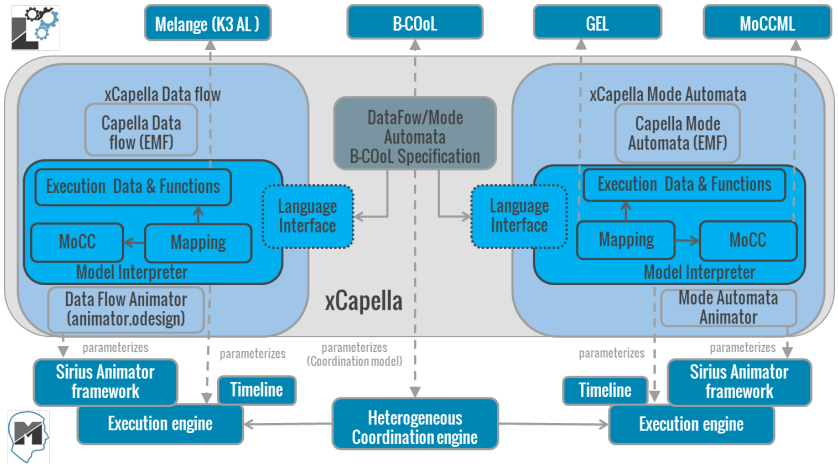

Fig. 1. xCapella

specification of the system to be built. Our goal is to reduce the risks concerning inconsistent functional requirements by providing a simulation environment of the existing specification, suitable to understand/analyse the system behavior.

3.2 Current experimentation

This section presents our approach to design the concurrency-aware xDSMLs of Capella. This experiment relies on the Capella metamodel (which is publicly available[9]) augmented with a dedicated extension for mode automata. This mode automata extension has been done by using Kitalpha, integrated to the Capella studio.

The GEMOC language workbench

Definition of behavioral semantics: To provide a behavioral semantics, we defined the semantics in two steps: (1) an extension of the metamodel with execution function and execution data and (2) the concurrent control flow definition. In this section, we focus on the definition of the mode automata semantics. The data flows semantics is not shown in this article but is used in the coordination pattern specification defined later in this paper. The mode automata DSML (Fig. 2 at the left side) has been extended with kitAlpha in order to add classes, attributes, references specifying the Execution Data (ED) (Fig. 2 at the right side). With the *Melange* tooling support, the mode automata DSML is extended by the definition of the execution functions, which define the sequential part of the mode automata operational semantics. Melange weaves the additional operation implementations specifying the execution functions (Fig. 2 at the bottom side).

[9] `https://www.polarsys.org/projects/polarsys.capella`

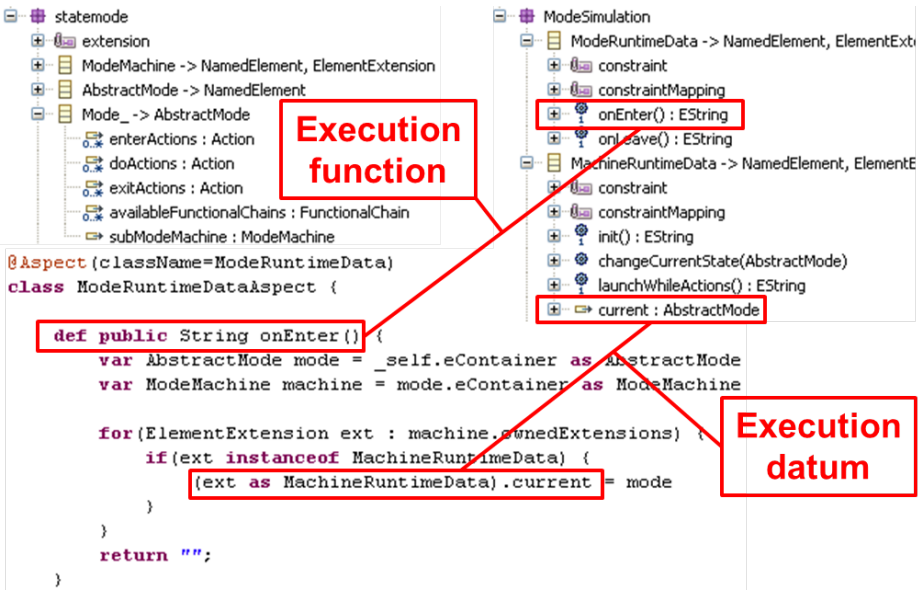

Fig. 2. GEMOC-xCapella: Definition of execution functions and data

The execution functions are orchestrated by the definition of a data-independent concurrent control flow (the data-dependent aspects of the control flow are encapsulated in the execution functions). In our approach, this control flow is captured in the so called Model of Concurrency and Communication (MoCC). The MoCC is a set of DSEs, specifying at the language level how, for a specific model, the event structure defining its concurrent control flow is obtained. The event structure represents all the possible execution paths of the model (including all possible interleavings of events occurring concurrently). For the definition of this control flow we used MoCCML [7] to specify our MoCC. MoCCML is a declarative meta-language designed to express constraints between events. The constraints can be capitalized into some libraries that are agnostic of any abstract syntax. The MoCC is compiled to a Clock Constraint Specification Language (CCSL) model interpreted by the TimeSquare tool [9]. The definition of the DSEs is realized by using the Event Constraint Language (ECL [8]), an extension of OCL which allows the definition of DSE in the context of concepts from the metamodel (see listing 1.1 where DSEs *entering* and *leaving* are defined in the context of an *AbstractMode*). Finally, the behavioral semantics is obtained by using a *Communication Protocol* which maps some of the DSEs from the MoCC to the execution functions (see Listing 1.1 where the DSEs are mapped to the execution functions *onEnter()* and *onLeave()* defined in the extension of Figure 2). This means that at the model level, when an event occurs, it triggers the execution of the associated execution function on an element of the model. Currently the implemented communication protocol is quite simple

but *GEL* [12] can be used to support more complex communication, for instance to specify data-dependent control.

Listing 1.1. Partial ECL specification of the mode automata

```
package statemode
 context AbstractMode
  def : entering : Event = self.ownedExtensions->select(E |
                                E.oclIsTypeOf(ModeRuntimeData))->first().
                                onEnter()
  def : leaving : Event = self.ownedExtensions->select(E |
                                E.oclIsTypeOf(ModeRuntimeData))->first().
                                onLeave()
```

Definition of the animation layer: We provide a new Sirius specification model (animator.odesign) which defines how the model representations change during the simulation. The animator is an extension of the concrete syntax definition which is part of Capella contributing a xCapella animation layer(Fig. 3) which customizes shape styles to highlight activated transition, add a decorator for the current mode and declare actions to toggle breakpoints.

The Sirius Animator framework also bring an integration with the Eclipse Debug user interface to inspect the runtime state of the execution, navigate to the corresponding diagrams and control the execution step by step.

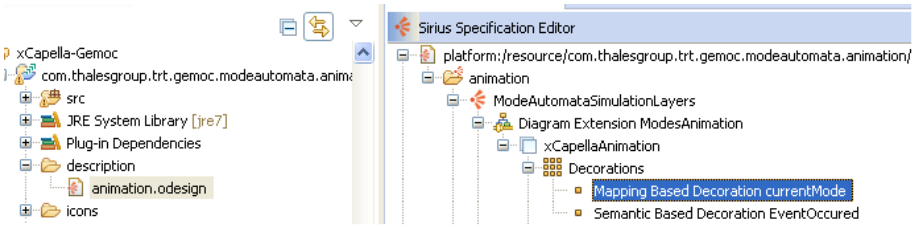

Fig. 3. GEMOC-xCapella animation layer

Definition of the coordination between xData-Flow and the xMode Automata: Once both the xData-Flow and the xMode Automata have been independently developed, it is of prime importance to specify the interactions of their models. This is realized by the specification of behavioural coordination pattern in BCOoL (Behavioral Coordination Operator Language).

In our case, a mode is associated to some functional chains. The coordination pattern must specify that a functional chain is activated (but not necessarily started) when the mode automata is in a specific mode. Consequently, when a mode automata is in a specific mode, the functional chains not associated to this mode are deactivated.

The BCOoL behavioral pattern contains two parts:

– a *matching*, which defines a predicate based on the DSE context to identify what are the events to coordinate in a specific model; and

– a *MoCCML constraint*, to specify how the matched events are coordinated.

In our BCOoL specification (see listing 1.2), the *ModeEnteringActivateFunctionalChain* operator coordinates the action of entering and leaving a mode with the activation of a functionalChain. Entering into a mode is identified by the entering DSE defined in the context of an AbstractMode in the *mode automa behavior language interface* (*i.e.*, in modemachine.ecl). Instances of such DSE have to be coordinated with instances of the *activate* DSE defined in the *data flow behavior language interface* (*i.e.*, CapellaDataflow.ecl). The *matching* specifies that the *entering* and *leaving* event from a mode are coordinated with the *activate* event from the functional chain only if the functional chain is referenced by the mode (in the *availableFunctionalChains* collection).

Listing 1.2. Heterogeneous coordination operator between the data flow and mode automata languages

```
BCOoLSpec  XCapellaDataFlow-xCapellaModeAutomata

ImportLib 'platform:/plugin/org.gemoc.xcapella.coordination/constraint/
    modeAutomata.moccml'

ImportInterface 'platform:/plugin/org.gemoc.xcapella.dataflow.dse/ecl/
    CapellaDataflow.ecl' as dataflow
ImportInterface 'platform:/plugin/com.thalesgroup.trt.mocc.modemachine.dse/
    ecl/modemachine.ecl' as modeautomata

Operator ModeEnteringActivateFunctionalChain (enter: statemode::entering,
    leave: statemode::leaving, activate: fa::activate)
    When:
      enter.availableFunctionalChains->exists(fc | fc = activate)
    CoordinationRule:
      enableElementWhenCurrentMode(activate, enter, leave)
end Operator;
```

The GEMOC-xCapella modeling workbench Once the xDSMLs implemented with the aforementioned tools of the language workbench, they are automatically deployed into the original *Capella* modeling workbench (integrated with the *GEMOC modeling workbench*). It results in an advanced modeling workbench integrated into the Eclipse debugger for model execution. The GEMOC-xCapella modeling workbench (Fig. 4) offers an environment for system engineers to understand/control the execution of their models with :

1. a graphical feedback of their model execution. For instance, in Figure 4, the green arrow on *initializeSystem* state represents the current state.
2. a possibility to explore several execution traces with a graphical timeline that supports step forward and step backward. The timeline and the concurrent logical step decider can be used conjointly by a designer to choose the next step in case of non determinism or concurrent events. For instance in the timeline, each vertical list of bullets represents some possible futures at this step. Also, at any time during the simulation, the designer can go back in the past to explore an alternative future.
3. a possibility to add some breakpoints to *pause* the simulation when the element carrying the breakpoint is touched (*i.e.*, when an operation is called on it).

Additionnaly, a designer can use the *execution model*, which represents the causalities and synchronizations in the model (*i.e.*, the timemodel file) to generate the state space of all possible execution traces from the concurrency point of view.

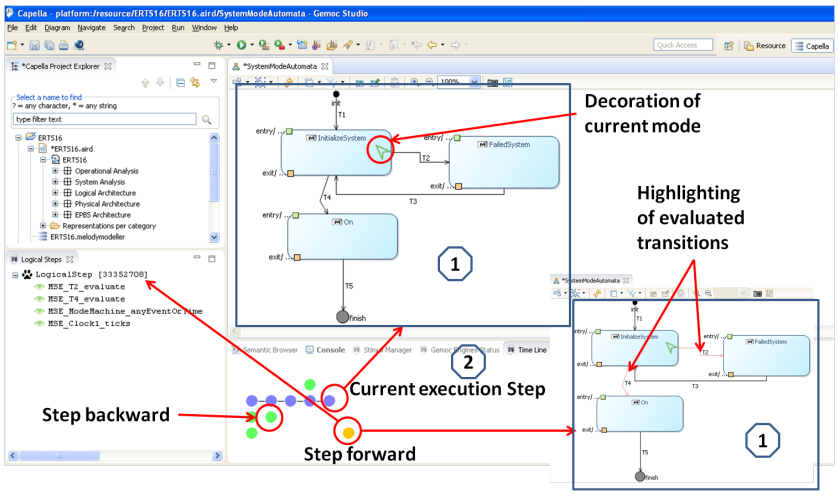

Fig. 4. GEMOC-xCapella modeling workbench

4 Related Works

In the past few years, some approaches proposed to specify the execution semantics of DSMLs by using fUML [14, 15]. While these approaches take good care to separate the execution semantics from the abstract syntax of a language, they specified the behavioral semantics as a whole by using fUML. In our approach, we use an explicit MoCC, execution functions and a protocol between them. It allows reasoning explicitly on the concurrency aspect of a language (data independently) but more important the protocol provides a natural language interface on which coordination patterns can be specified to automatically obtain coordinated simulation of heterogeneous models.

Ptolemy [11] and Modhel'x [3] also provide capabilities to simulate coordinated heterogeneous models but compared to our approach, the associated framework neither rely on a user defined abstract syntax nor on explicit coordination patterns, amenable to the easy customization of the coordination to fit the domain of use.

Finally, when some models are coordinated with our approach, it relies on both an explicit behavioral semantics and an explicit coordination. Making explicit the behavioral semantics and the coordination enables the comprehensive incorporation of semantic adaptation between the heterogeneous compo-

nent. This is a major difference compared to existing approaches based on co-simulation bus (e.g. where they use the FMI/FMU standard[10]) in which the coordination is either done in the importing tool or by the manual writing of a master on the bus [2, 13]. Co-simulation bus approaches are very complementary to our approach. We believe that our approach can be used earlier in the development process, to allow, for instance synthesizing a bus master according to the explicit specification of the coordination.

5 Conclusion and perspectives

The GEMOC methods and tools have been validated through the use of an experimental (Technology Readiness Level 3) integrated advanced simulation prototype (Fig. 4). The experiment is focused on: the use of the GEMOC methodology and studio to define the behavioral semantics and coordination of *mode automata* and *data flow*; the customization of each language graphical notation for animation. The GEMOC modeling workbench provides also a well-integrated model debugging environment based on Eclipse, including advanced features for graphical model animation and execution trace management (time line). Finally, we have a proof of concept of the integration of the GEMOC execution engine and the Sirius animator framework into the Capella legacy industrial engineering workbench. The experiments result in a prototype named *xCapella*, an extension of Capella that supports the execution and animation of behavioral models. For now, even if the coordination pattern between data flows and mode automata languages has been defined, the GEMOC *heterogeneous coordination engine* [16] is not integrated to xCapella; this task is already started. Some longer terms perspectives are the definition of the behavioral semantics of the Capella scenario language and to the identification of xCapella main semantics variation points. It is also planed to provide an export of an executable model in the FMI2 standard[11].

Acknowledgement. This work is supported by the ANR INS Project GEMOC (ANR-12-INSE-0011) and The GEMOC Initiative (cf. `http://gemoc.org/ins`).

References

1. Systems and software engineering – architecture description. ISO/IEC/IEEE 42010:2011(E) pp. 1–46 (2011)
2. Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauß, C., Elmqvist, H., Junghanns, A., Mauss, J., Monteiro, M., Neidhold, T., et al.: The functional mockup interface for tool independent exchange of simulation models. In: 8th International Modelica Conference, Dresden. pp. 20–22 (2011)
3. Boulanger, F., Hardebolle, C.: Simulation of Multi-Formalism Models with ModHel'X. In: Proceedings of ICST'08. pp. 318–327. IEEE Comp. Soc. (2008)

[10] http://fmi-standard.org
[11] `www.fmi-standard.org`

4. Combemale, B., Deantoni, J., Baudry, B., France, R., Jézéquel, J.M., Gray, J.: Globalizing Modeling Languages. Computer pp. 68–71 (Jun 2014), http://hal.inria.fr/hal-00994551
5. Combemale, B., Deantoni, J., Vara Larsen, M., Mallet, F., Barais, O., Baudry, B., France, R.: Reifying Concurrency for Executable Metamodeling. In: Martin Erwig, R.F.P., van Wyk, E. (eds.) 6th International Conference on Software Language Engineering (SLE 2013). Lecture Notes in Computer Science, Springer-Verlag, Indianapolis, 'Etats-Unis (2013), http://hal.inria.fr/hal-00850770
6. Deantoni, J., Issa Diallo, P., Teodorov, C., Champeau, J., Combemale, B.: Towards a Meta-Language for the Concurrency Concern in DSLs. In: Design, Automation and Test in Europe Conference and Exhibition (DATE). Grenoble, France (Mar 2015), https://hal.inria.fr/hal-01087442
7. Deantoni, J., Issa Diallo, P., Teodorov, C., Champeau, J., Combemale, B.: Towards a Meta-Language for the Concurrency Concern in DSLs. In: Design, Automation and Test in Europe Conference and Exhibition (DATE'15). Grenoble, France (Mar 2015), https://hal.inria.fr/hal-01087442
8. Deantoni, J., Mallet, F.: ECL: the Event Constraint Language, an Extension of OCL with Events. Research Report RR-8031, INRIA (Jul 2012), https://hal.inria.fr/hal-00721169
9. DeAntoni, J., Mallet, F.: TimeSquare: Treat your Models with Logical Time. In: 50th Int. Conf. on Objects, Models, Components, Patterns. LNCS, vol. 7304, pp. 34–41. Springer (2012)
10. Degueule, T., Combemale, B., Blouin, A., Barais, O., Jézéquel, J.M.: Melange: A Meta-language for Modular and Reusable Development of DSLs. In: 8th International Conference on Software Language Engineering (SLE). Pittsburgh, United States (Oct 2015), https://hal.inria.fr/hal-01197038
11. Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity – the Ptolemy approach. Proc. of the IEEE 91(1), 127–144 (2003)
12. Latombe, F., Crégut, X., Combemale, B., Deantoni, J., Pantel, M.: Weaving Concurrency in eXecutable Domain-Specific Modeling Languages. In: 8th ACM SIGPLAN International Conference on Software Language Engineering (SLE). ACM, Pittsburg, United States (2015), https://hal.inria.fr/hal-01185911
13. Marinescu, R., Kaijser, H., Mikučionis, M., Seceleanu, C., Lönn, H., David, A.: Analyzing industrial architectural models by simulation and model-checking. In: Artho, C., Ölveczky, P.C. (eds.) Formal Techniques for Safety-Critical Systems, Communications in Computer and Information Science, vol. 476, pp. 189–205. Springer International Publishing (2015), http://dx.doi.org/10.1007/978-3-319-17581-2_13
14. Mayerhofer, T., Langer, P., Wimmer, M., Kappel, G.: xMOF: Executable DSMLs based on fUML. In: 6th Int. Conf. on Software Language Engineering. LNCS, vol. 8225, pp. 56–75. Springer (2013)
15. Tatibouët, J., Cuccuru, A., Gérard, S., Terrier, F.: Formalizing Execution Semantics of UML Profiles with fUML Models. In: 17th International Conference on Model Driven Engineering Languages and Systems. LNCS, vol. 8767, pp. 133–148 (2014)
16. Vara Larsen, M., Deantoni, J., Combemale, B., Mallet, F.: A Model-Driven Based Environment for Automatic Model Coordination. In: CEUR (ed.) Models 2015 demo and posters. Models 2015 demo and posters, Ottawa, Canada (Oct 2015), https://hal.inria.fr/hal-01198744
17. Vara Larsen, M.E., Deantoni, J., Combemale, B., Mallet, F.: A Behavioral Coordination Operator Language (BCOoL). In: Lethbridge, T., Cabot, J., Egyed, A. (eds.) International Conference on Model Driven Engineering Languages and Systems (MODELS). p. 462. No. 18, ACM, Ottawa, Canada (Sep 2015), https://hal.inria.fr/hal-01182773