Towards Executable UML Interactions based on fUML

Marc-Florian Wendland
Fraunhofer Institut FOKUS, Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany

Keywords: UML Interactions, fUML, UML Activities, Executable UML, Executable Interactions, PSCS.

Abstract: Executable specifications for UML currently comprise fUML, precise semantics of composite structures and in future precise semantics for state machines. An executable semantics for UML Interactions is on the roadmap, but has not been addressed by the OMG Executable UML working group so far. Interactions are said to be the second most used diagrams after class diagram of UML, thanks to their comprehensibility and illustrative visualization. Unfortunately, they suffer from fuzzy semantics and technical issues that wastes the potential Interactions could have for engineering activities apart from high-level specifications. In this position paper we present first results from experiments and attempts to map UML Interactions to fUML Activities in order to eventually execute them.

1 INTRODUCTION

UML Interactions, better known as sequence diagrams, are a one of the standardized UML (UML, 2015) behavior kinds that describe the exchange of messages among parts of a system or sub-system, represented by Lifelines. Interactions are familiar to many stakeholders because of their illustrative graphical notation that is easy to comprehend. However, the UML Interactions metamodel is not as precise as desired, in particular with respect to data (Wendland et al., 2013). As a result, they are mostly used for sketching high-level specifications of use cases or early protocol specification. UML Interactions reveal global viewpoint on the interplay of participating instances. This is different to UML State Machines or UML Activities that usually describe the behavior of dedicated participants from an internal (or local) point of view. In fact, UML Interactions are the only standardized UML behavior that describe the exchange of Messages from a global point of view. This give rise to the fact that Interactions usually do not have no direct counterpart in an implementation. They rather resembles a virtual window that allows insights into the execution of system at a certain point of time.

UML Interactions are said to be the UML's second most used diagrams (after class diagrams) due to their easily comprehensible notation that foster communication among stakeholder. For a more precise system specification, however, the aforementioned fuzzy semantics of UML Interactions (including but not limited to the lack of data flow concepts) makes it complicated to exploit their whole potential. Executable Interactions help to overcome the semantical shortcomings by explicitly stating which features of UML Interactions can be used for precise (yet executable) specifications and how these features are modelled best with the UML Interactions metamodel.

A working group at OMG has started giving subsets of the UML behaviors a precise yet executable semantics. The first standard of executable UML was released in 2011 based on UML Activities. It is called *Foundational Subset for Executable UML Models* (fUML, 2012), in short fUML. Meanwhile, a precise specification of the semantics of composite structures (PSCS, 2015) is on the verge of being standardized and above all the executable UML working group is currently heading towards precise semantics for executable state machines (PSSM). UML Interactions are already on the roadmap of the Executable UML working group but not in focus yet.

In this position paper, we report first results of a mapping from UML Interactions to fUML in order to assign the UML Interaction building blocks an operational semantics. This enables the use of Interactions for building executable specification, which has, as we are certain, an enormous potential to improve the entire development chain, spanning from requirements engineering, rapid prototyping to testing of executable specifications.

We focus in this position paper on UML

Wendland, M-F.
Towards Executable UML Interactions based on fUML.
DOI: 10.5220/0005809804050411
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 405-411
ISBN: 978-989-758-168-7

Interactions in an isolated way. UML behaviors are, at least technically and theoretically, integrated with each other. This enables engineers to select the behavioral kind that is most appropriate for him or her. The main objective of our work is to provide an operational semantics for the constituents of an Interaction (e.g. Messages, CombinedFragments etc.) and the flow of execution within Interactions. As such, the discussion about how Interactions integrate with other UML behaviors (first and foremost fUML Activities) is not part of this paper, but is rather a research paper on its own.

This position paper implies familiarity with the UML metamodel, in particular with Interactions and Activities. Therefore, we spare an introduction into the semantics and the metamodel of UML Interactions, UML Activities and fUML.

The remainder of this paper is structured as follows: In Section 2 the work related to our work will be summarized. Section 3 summarizes the base semantics and preliminary mapping rules of our approach. Section 4 evaluates a proof-of-concept that demonstrates the feasibility of our work. It summarizes a issues we faced when working on the mapping. Section 6 eventually concludes this paper and sketches future work in the area of executable UML Interactions.

2 RELATED WORK

A lot of work has been done in the first few years of UML 2 in the realm of formal semantics of UML Interaction traces. Haugen compares UML Interactions and Message Sequence Charts (Haugen, 2004) showing that Interactions and MSCs are similar down to small details.

Haugen, Stolen, Husa, and Runde have written a series of paper on the compositional development of UML Interactions supporting the specification of mandatory and potential behavior, called STAIRS approach (Haugen and Stølen, 2003; Haugen, Husa, Runde and Stølen, 2005; Haugen, Husa, Runde and Stølen, 2005; Runde, Husa, Haugen and Stølen, 2005). A depper analysis is dedicated to a fine-grained differentiation of event reception, consumption and timing (Haugen, Husa, Runde and Stølen, 2005) and the refinement of Interactions with regard to underspecification and nondeterminism (Runde, Husa, Haugen and Stølen, 2005). Lund and Stolen have presented an operational semantics for UML sequence diagrams (Lund and Stølen, 2003).

Formal semantics of UML Interactions and sequence diagrams were several times discussed. Störrle presented a formal specification of UML Interactions and a comparison of UML 2.0 and UML 1.4 Interactions (Störrle, 2003; Störrle, 2004). A similar work was done by Knapp and Cengarle (Knapp, 1999; Cengarle and Knapp, 2004), Li and Ruan (Li and Ruan, 2011) and Shen et al., (2008). Special attention was set to the semantics of assert and negative CombinedFragments (Störrle, 2003; Harel and Maoz, 2006), though. Prior to UML sequence diagrams Damm and Harel (Damm and Harel, 1999) worked on a notation called Life Sequence Charts.

The work done by Wendland et al., (2013) focuses a different aspect of UML Interactions, namely the precise definition of Message arguments. In that case, their work is different to the previously mentioned papers that mostly dedicated to the trace semantics of Message reception and consumption within UML Interactions, but they did not focus on precisely specifying data transmitted by Messages.

The work described in this paper is different to all the previously mentioned work for it uses fUML to state the semantics of UML Interactions. fUML's formal semantics was defined using the ISO standard *Process Specification Language* (PSL, 2004) and Common Logic (CL, 2007). The decision to map UML Interactions to fUML was made to provide engineers with an easy to understand and familiar notation (UML Activities) for executable Interactions. We think this increase both the comprehensibility and applicability of executable Interactions. In fact, an engineer who knows UML Activities and its action semantics is capable of reading and writing UML Interactions in a precise yet executable manner.

3 TOWARDS EXECUTABLE INTERACTIONS

The following section discusses some of the fundamental concepts for executable Interactions. Therefore, we related the semantics of UML Interactions and some of its building blocks to fUML concepts. Besides technical discussions we also highlight restrictions to some meta-concepts of such as the binding character or general trace semantics of UML Interactions.

3.1 Descriptive Vs Prescriptive Behaviors

UML introduces the notion of *descriptive* versus

prescriptive behaviors. UML State Machines and Activities are prescriptive, whereas Interactions are by definition descriptive. As such, Interactions tell only parts of the story that is actually going on at runtime. They do not claim to be complete, although it not prohibited interpreting Interactions in such a way. Descriptive (or partial) behavior is in particular appropriate on a higher specification level, but for model execution, prescriptive behavior is required. Hence, the fundamental difference between executable and non-executable Interactions is that the first one prescribes explicitly what *will* happen and does not describe what *may* happen at runtime.

3.2 Invalid Vs Valid Traces

UML Interactions are the only behavior that differentiates between invalid and valid occurrence traces, though. It enables engineers to model scenarios that represent invalid system behavior, or unwanted scenarios. Invalid traces are predestined means to specify unwanted or undesired scenarios, however, there is no understanding of the consequences an invalid behavior entails once occurred. This underpins the higher-level specification character of Interactions, though. Invalid traces are usually resolved into concrete measures of an implementation or simulation like e.g., throwing an exception or any other adequate measure to handle and mitigate invalid system or environment behavior (e.g., a user or an interacting system behaves incorrectly). In fact, resolved invalid traces constitute valid traces that specify what shall happen in erroneous situations. Therefore, the notion of invalid traces is excluded for executable Interactions.

3.3 Ordering of Execution Traces

The main building blocks an Interactions consists of are occurrences. An occurrence in the realm of Interactions is the smallest piece of executable behavior (e.g. a statement) and manifests as instance of the metaclass OccurrenceSpecification in the metamodel. (Actually, the real building blocks are InteractionFragments, but only OccurrenceSpecification represent execution of event.) An OccurrenceSpecification is, thus, semantically close to the metaclass Action that represents the fundamental building blocks of Activities.

fUML specifies the execution of a behavior as a set of event occurrences called *execution trace* that span from the invocation of the behavior over behavior-local event occurrences to its termination. This matches very well with the trace semantics of Interactions that orders the InteractionFragments it contains both globally and locally. The local InteractionFragments (i.e., those covering the very same Lifeline) result in corresponding Actions as owned behavior of the Class that is represented by the Lifeline. The corresponding Actions are connected by ControlFlow edges (remember, no data manipulation concepts). Global ordering of InteractionFragments is for the construction of an Activity from a Lifeline less important. If all the Lifelines are properly mapped to Activities, abidance of global event occurrence ordering is a result of abidance of local even occurrence ordering.

The concrete Actions that constitute the resulting executable fUML behavior of an Interaction according to the covering InteractionFragments depends eventually on the mapping rules for those InteractionFragments.

3.4 Interaction and Lifelines

UML Interactions represent a global view on the interplay of system parts. Even though we said earlier that executable Interactions have to be prescriptive, the global viewpoint of Interactions holds still true. Nonetheless, the handling of a Lifeline's lifecycle needs to be co-ordinated. Therefore, an Interaction (Interaction is also the name of the metaclass that contains all building block, thus, it builds the outermost boundary of the behavioral description) is mapped to an Activity. This Activity is then responsible for co-ordination. The co-ordinating (henceforth called *main*) Activity is responsible to initially create, accept (in case of an invocation of an Interaction) and finally destroy Lifelines. The semantics of Lifeline handling shall be aligned with the PSCS semantics (PSCS, 2015), in particular with its instantiation patterns.

A Lifeline indirectly represents (by representing a part in a composite structure) a set of instances of a Classifier. This Classifier shall only be of type Class (there are further concrete subclasses of Classifier such as DataType or Collaboration, though). The reason is that only instances of the UML metaclass BehavioralClassifiers are allowed owning behaviors and as such offering methods to its environment.

Even though possible, executable Interactions restrict the number of Lifelines that represent the same part (role of a Class in the underlying composite structure) to exactly one. This is mainly because the selection mechanism of Lifelines, that distinguishes sets of instances of the same role, is not precisely

specified in UML and requires clarification first. UML states that the ValueSpecification that constitute the selection mechanism shall evaluate to a positive (range of) Integer, identifying the respective instances in the set of all instances of the role. Since roles may represent unordered collections, selection by index cannot be applied.

Due to the global viewpoint of UML Interactions, the use of parameters and global attributes needs careful treatment. Such global ConnectableElements (ConnectableElement is the common superclass of Parameter and Property in UML), are in fact shared among all Lifelines that take part in an Interaction and, thus, need to be accessible by all Lifelines. This means that there must be coordinating instance that holds the ConnectableElements and grants access to them. Since Interactions provide no concepts for data flows or modifications as described in clause 17.1.1 in UML 2.5 ("…*but the Interactions do not focus on the manipulation of data even though data can be used to decorate the diagrams*."), the use of global ConnectableElements is discouraged for the time being. Wendland, Schneider and Haugen have recognized the lack of concepts to describe data flows in Interactions and have provided a minimal extension to the UML metamodel to mitigate these shortcomings. This means that at some point in future, when data flows are supported by Interactions, the use of global ConnectableElements will be, for sure, reconsidered by our work.

3.5 Mapping Rules Overview

Based on first investigations and experiments with executable (i.e., fUML compliant) Interactions, we identified a set of mapping rules between Interactions and fUML metamodel. We spared all UML Interaction concepts we have not yet fully investigated. This means, every metaclass that is not mentioned in Table 1 is left open for future work.

4 PROOF-OF-CONCEPT

Several prototypic (manual) compilations from UML Interactions to fUML-based executable Interactions have been performed prior to this work. A simple abstract example that illustrates the feasibility of our idea is depicted in Figure 1.

Table 1: Preliminary mapping rules for UML Interactions.

UML Interaction concept	fUML Activity concept
Interaction	(main) Activity
Lifeline	Class with classifierBehavior set to the mapped Activity
Sending MessageOccurrence Specification (MOS)	SendSignalAction (later also CallOperationAction, Create/DestroyObjectAction)
Receiving MOS	AcceptEventAction (or request to create/destroy instances)
Message	Arguments and signature used to complete InvocationAction; mapped solely to Signal sending and reception.
CombinedFragment (only par, loop, alt, seq, strict)	LoopNode, DecisionNode, ForkNode, ConditionalNode
GeneralOrdering	Indirectly mapped to co-ordinating Signals
InteractionOperand	Sequence of Actions connected by ControlFlow

4.1 Descriptions of the Example

The Collaboration *ServiceChoreography* consists of two parts that are connected via a Connector that ends in compatible Ports. The Interaction *SD1* (potentially in addition to further Interactions of the Collaboration) is visualized as sequence diagram in the lower compartment of the Collaboration (which serves mere illustration purposes, but is not standard UML). The Collaboration, its composite structure and its owned behavior *SD1* are compiled into a corresponding fUML- and PSCS-compliant model afterwerds.

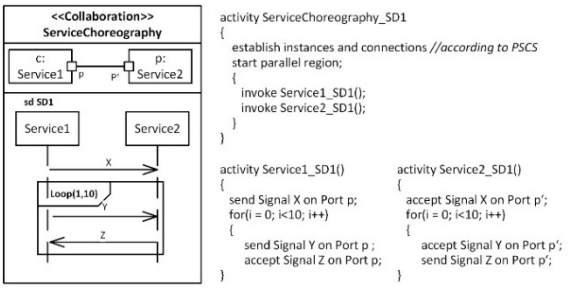

Figure 1: Early proof-of-concepts for executable Interactions.

In this example, however, we will solely focus on the mapping of behavioral aspects. A precise integration with PSCS, especially the create of new

instances (respectively Lifelines) is not covered in this position paper.

The main Activity *ServiceChoreography_SD1* is responsible to instantiate and connect the instances according to the PSCS rules. Afterwards, the Operations that represent the respective Lifeline behavior is started in parallel. Due to the already instantiated and working instances, events can be put into the respective event pools, thus, events will not get lost, even if *Service1_SD1* sends the *Signal X* before *Service2_SD1* awaits the Signal. The methods of *Service1_SD1* and *Service2_SD1* are written in pseudo-code that resemble literally the corresponding fUML actions. We thought about employing the *Action Language for Foundational UML* (Alf, 2013) to denote the resulting fUML, but found the pseudo-code simpler to understand for non-Alf experts, though.

4.2 Evaluation

The proof-of-concept has confirmed the general applicability of our idea. We were able to (manually) translate the UML Interaction to a fUML Activity and execute the Activity afterwards.

However, we spared on purpose more complex aspects such as how Interactions integrate with other behaviors and composite structures, and more complicated building blocks (e.g., guards on CombinedFragements, nested CombinedFragments, Gates, creation/deletion of Lifelines etc.). These aspects are subject to future research.

4.3 Faced Issues

The most severe fUML issue we encountered was the missing of CallEvents and corresponding Actions like AcceptCallEvent etc. Additionally, a CallOperationAction is currently only allowed to be synchronous, which is not sufficient for Interactions. The lack of CallEvents leads to a situation where a Lifeline may only actively wait for SignalEvents, but not for CallEvents. Besides, the ongoing work on the precise semantics for executable state machines already identified the need to treat Signal sending and Operation invocation equally based on the event handing mechanism. There has already been an issue submitted to OMG for this. We overcame this shortcoming by mapping Messages with MessageKind set to asynchCall and synchCall to SendSignalActions and AcceptEventActions. For future versions of fUML, however, we hope that the missing Actions are incorporated into the language. The fact that the work on PSSM introduces CallEvents, we are confident that they will become part of fUML as well rather soon.

Another fUML issue we found is that it does not support the concept of owned behaviors unless they represent the method of an Operation. This excludes the BehaviorExecutionSpecification from the mapping for it does not have an added value anymore. Since it is not possible to pass parameters from the calling Lifeline to an Behavior referenced from a BehaviorExecutionSpecification and a Behavior cannot be stand-alone owned by a Class, the only behavior that could invoked by BehaviorExecutionSpecifications is a context-free Activity with optional or no parameters at all. Such an invocation is of very limited use. We propose an improvement to fUML to allow also context-aware, stand-alone owned behaviors as well as a solution to pass Parameters into BehaviorExecutionSpecifications.

A third big, yet UML Interactions issue that needs attention is the lack of concepts for describing data flows. We hope that this issue will be eventually addressed by executable UML state machine where it is also required to access the data received by an InvocationEvent for further processing (e.g., in Transition guards and effects). Without data flow mechanism, the use of Interaction for execution is limited to the simulation of high- level specifications where arguments of Messages have to be provide always a priori. Such simulations may have their use, but waste a lot of potential. We argue for addressing the lack of data flow concepts in UML Interactions rather sooner than later.

Apart from the issues that belong to fUML or UML itself, there are a number of unresolved or unclear mapping rules in our approach that need further investigation. In this position paper, however, we only discussed the mapping rules we had tried out and verified so far. The UML Interactions metamodel offers further metaclasses (such as Gate, PartDecomposition, creation Message) that we have not yet addressed.

5 CONCLUSIONS

In this position paper, we reported first results of our preliminary work in the realm of executable UML Interactions by relying on fUML semantics. We argued why it make sense to strive for executable Interactions, represented first and foremost by sequence diagrams. We discussed the necessity for the shift from descriptive UML Interactions to prescriptive executable Interactions. Based on this,

we described a set of fundamental mapping rules from the Interaction metamodel to the fUML metamodel. A simple example was presented for which we depicted a pseudo fUML snippet. Finally, we raised awareness for open issues in fUML and UML that we faced during our (manual) translation. This approach is, to the best of our knowledge, the first attempt to translate Interactions to fUML.

One surprising finding is that Interactions and Activities, apart from fundamentally different building blocks, are actually quite close to each other. Even if not reported in this position paper, we have identified suitable mappings for almost all concepts in Interactions. Some of those mappings (which we spared in this paper) are based on the assumption that fUML supports the execution of context-aware owned behaviors and CallEvents.

In particular the seamless integration of executable Interactions with other executable UML behaviors and the precise semantics of composite structures needs more attention. For the sake of simplicity, we treated Interactions in an isolated way in our work. This led to a working, but autarkic proof-of-concept. Such an autarkic view is suitable in order to focus on the executable semantics of building blocks firstly, but for a realistic application of executable specifications, the seamless integration needs to be achieved. Rules and constraints have to be identified and specified to assist engineers building such seamless and interworking executable specification that potentially consist of fUML, executable state machines, executable Interactions and precise composite structures.

Future work in that area targets in particular completion of our mapping rules. We plan furthermore to support the executable UML working group at OMG in raising awareness of the issues we found and in resolving these issues. Our long-term goal, however, is the utilization of fUML for building a seamlessly integrated test execution system for fUML simulations. The upcoming OMG standard UML Testing Profile 2 enables specifying test case specifications as Interactions, which are compiled into executable test cases based fUML.

REFERENCES

Haugen, Ø. and Stølen, K.: STAIRS — Steps to analyze interactions with refinement semantics. In Proc. International Conference on UML, 2003.

Haugen, Ø., Husa, K. E., Runde, R. K., and Stølen, K.: Why timed sequence diagrams require three-event semantics. In Scenarios: Models, Transformations and Tools, 2005.

Haugen, Ø., Husa, K.E., Runde, R.K., and Stølen, K.: STAIRS towards formal design with sequence diagrams. Journal of Software and Systems Modeling, 2005.

Runde, R. K., Haugen, Ø., Stølen, K.: Refining UML interactions with underspecification and nondeterminism. In: Nordic Journal of Computing, 2005.

Lund, M. S., and Stølen, K.: A fully general operational semantics for UML 2.0 sequence diagrams with potential and mandatory choice. In: Proceedings of the 14th international conference on Formal Methods (FM'06), 2006.

Störrle, H.: Semantics of interactions in UML 2.0. In: Proceedings of IEEE Symposium on Human Centric Computing Languages and Environments, 2003.

Störrle, H.: Trace Semantics of UML 2.0 Interactions. Technical report, University of Munich, 2004.

Knapp, A.: A Formal Semantics for UML Interactions. In: R. France and B. Rumpe (eds.): Proc. 2nd Int. Conf. Unified Modeling Language (UML'99), 1999.

Cengarle, M., Knapp, A.: UML 2.0 Interactions: Semantics and Refinement. In: J. Jürjens, E. B. Fernàndez, R. France, B. Rumpe (eds.): 3rd Int. Workshop on Critical Systems Development with UML (CSDUML'04), 2004.

Li, M., and Ruan Y.: Approach to Formalizing UML Sequence Diagrams. In: Proc. 3rd In-ternational Workshop on Intelligent Systems and Applications (ISA), 2011.

Shen, H., Virani, A.; Niu, J.: Formalize UML 2 Sequence Diagrams. In: Proc. 11th IEEE High Assurance Systems Engineering Symposium (HASE), 2008.

Störrle, H.: Assert, Negate and Refinement in UML-22 Interactions. In: J. Jürjens, B. Rumpe, R. France, and E. B. Fernandez, Proc. Wsh. Critical Systems Development with UML (CSDUML'03), 2003.

Harel, D., and Maoz, S.: Assert and negate revisited: modal semantics for UML sequence diagrams. In: Proc. International workshop on Scenarios and state machines: models, algorithms, and tools, 2006.

Knapp, A., and Wuttke, J.: Model Checking of UML 2.0 Interactions. In; Proc. of the 2006 International conference on Models in Software Engineering (MoDELS'06), Springer, Heidelberg 2006.

Wendland, M.-F., Haugen, O., and Schneider, M.: Evolutions of UML Interactions metamodel. In; Proc. of the 2013 International conference on Models in Software Engineering (MoDELS'13), Springer, Heidelberg, 2013.

Damm, W., Harel, D.: LSCs: Breathing Life into Message Sequence Charts. J. on Formal Methods in System Design 19 (1), 45–80 (2001). In Proc. 3rd IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99), 1999.

UML, Object Management Group: Unified Modeling Language 2.5, http://www.omg.org/spec/UML, 2015.

fUML, Object Management Group: Semantics of a Foundational Subset for Executable UML 1.1, http://www.omg.org/spec/FUML/1.1, 2013.

PSCS, Object Management Group: Precise Semantics of Composite Structures, http://www.omg.org/spec/PSCS, 2015.

Alf, Object Management Group: Action Language for Foundational UML, http://www.omg.org/spec/ALF/, 2013.

CL, International Standards Organisation, Common Logic (CL): a Framework for a Family of Logic-Based Languages," ISO/IEC 24707:2007, 2007.

PSL, International Standards Organisation, Process specification language, ISO 18629-1:2004, 2004.