A Model-driven Approach to Process Enactment

Sana Damak Mallouli[1], Saïd Assar[2] and Carine Souveyet[1]
[1]*Centre de Recherche en Informatique, University of Paris 1, 90 rue Tolbiac, 75013 Paris, France*
[2]*Telecom Ecole de Management, 9, Rue C. Fourier, 91011, Evry, France*

Keywords: Meta-Modeling, Model-driven Engineering, Model Enactment, Event Modeling, Publish/Subscribe Pattern.

Abstract: Building software tools to support a new modeling formalism is a complex, error prone and time consuming task. Previous experiences have taught us that maintainability and portability are key issues which are poorly supported when development is realized in and ad-hoc manner. To overcome these limitations, we are investigating a meta-model driven approach for specifying at design phase not only the structural part of a process meta-model, but also its operational semantics in order to derive in a systematic manner an enactment engine. In this paper, we show how process model operational semantics are expressed by defining the architecture of an interactive enactment engine, and how the engine's behavior is formally specified using an event based notation. This approach includes an implementation step in which the engine behavior meta-model is transformed into a running system that is based on the publish/subscribe pattern.

1 INTRODUCTION

The topic of this paper is related to the construction of software tools that provide enactment mechanisms for process models. In previous works, our research team has engineered software tools for event-oriented and goal-oriented modeling formalisms (Rolland et al., 1988), (Souveyet and Tawbi, 1998). These tools were built in an ad-hoc manner, i.e. minimal specifications were provided at design time and project resources were, to a large extent, exclusively dedicated to development effort. These experiences have taught us that engineering such tools is complex, error prone and time consuming. Maintainability and portability are key issues, they are poorly supported as most of design knowledge is hard coded into the software program code. These tools neither lived beyond being temporary demonstration prototypes, nor could evolve to support any new modeling languages.

To overcome these drawbacks, we are investigating the application of model-driven engineering (MDE) approaches to software tools engineering. In MDE, models are first level artifacts through the whole software life-cycle. The final running system is obtained from design models through various transformation steps which can – when possible – be fully automated. This approach is expected to leverage designers and developers productivity, and to enhance product quality, maintainability and portability.

When building software tools according to MDE, meta-models become essential artifacts as they are expected to describe the structure, the semantics and the future usage of the models to be manipulated by the tool under construction. However, meta-modeling usage is generally restricted to specifying the static structure of models, i.e. concepts and links between these concepts (Sprinkle et al., 2011). While the *process* and the *behavior* perspectives are well known in information systems modeling (Olle et al., 1991), they are missing in meta-models specified in the software engineering domain. For a process modeling language, this knowledge inquires on its operational semantics.

A first study on the relationship between a meta-model and the expression of the underlying model's operational semantics was made on Petri nets in an early work by Breton and Bézivin (2001). The authors complemented the static meta-model describing the structure a Petri net (arcs and transitions) by a behavior meta-model which add necessary data structures for the execution of an instance of this model (tags and token movements). Out of these preliminary reflections, the Kermeta language was proposed and developed. Kermeta is an object-oriented meta-programming language. It provides a way to add meta-specification to an UML

Damak Mallouli S., Assar S. and Souveyet C..
A Model-driven Approach to Process Enactment.
DOI: 10.5220/0004083303510354
In *Proceedings of the 7th International Conference on Software Paradigm Trends* (ICSOFT-2012), pages 351-354
ISBN: 978-989-8565-19-8

meta-diagram (Kermeta, 2012).

Our work adopts a similar approach. We investigate how to specify at a high level of abstraction and using meta-models, both the structure and the semantics of process models, and how running software tools can systemtically or automatically be derived from such specifications. In a previous work (Assar et al., 2011), we analyzed the need for behaviour perspective in meta-modeling. In this paper, we refine this proposal and claim that expressing a process model operational semantics using a behaviour metamodel corresponds in fact to specifying an enactement engine. In order to derive a running software tool, we investigate the transformation of the behaviour meta-model into an executable object oriented application based on the publish/subscribe pattern.

Our research work focuses on the Map formalism, a goal-oriented modeling notation which is particularly well suited to represent engineering processes (Rolland et al., 1999). Earlier works have explored the possibility of building software tools dedicated to the Map formalism (Velez, 2003), (Edme, 2005). These exploratory prototypes were specified with structure oriented meta-models, and then developed in ad-hoc manners using programming languages and relational DBMS.

This paper is organized into 5 sections. Section 2 describes briefly Map modeling notation together with static meta-models. Section 3 introduce the behavior modeling notation and the behavior meta-model for the enactment engine. Section 4 presents the publish/subscribe pattern and defines transformation rules for deriving enactment tool. The last section discusses the proposed approach, and proceeds with the conclusion.

2 THE MAP FORMALISM

Based on the intention paradigm, the Map formalism captures the goals (Intentions) that a process is expected to fulfill, together with a set of available strategies to realize these intentions. Each *Intention* can be realized by one or more *Strategy*, and the whole process is represented as a labeled graph (Fig. 1a) with intentions as nodes and strategies as edges (Rolland et al., 1999). The operational semantics of the Map do not constrain the user in a sequential process consisting of successive steps, but allows instead a large degree of freedom in the scheduling of intentions and in the choice of the strategy to be applied at each step.

To be expressed precisely, Map operational

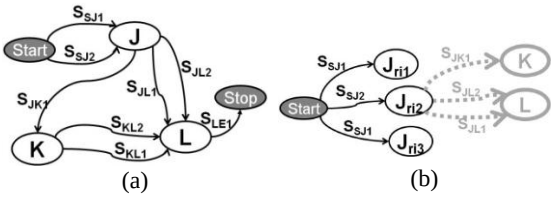

Figure 1: (a) A map illustrating example, and (b) an illustration of achieved intentions and candidate sections.

semantics need references to the product. Indeed, the achievement of an intention corresponds to the execution of some action on an instance of a product fragment according to a certain strategy. By executing a <J, S, L> section, we mean achieving the intention L, using the strategy S, and starting from the result (i.e. an instance of a product fragment) of intention J. If J_{ri1}, J_{ri2} and J_{ri3} are past *realizations* of intention J with different instances of the same product fragment (Fig. 1b), the sections <J, S_{JK1}, K>, <J, S_{JL1}, L> or <J, S_{JL2}, L> can then be executed for *each* of these past realizations. For the realized intention J_{ri2} for example, three sections are candidates for execution: < J_{ri2}, S_{JK1}, K>, < J_{ri2}, S_{JL1}, L> or < J_{ri2}, S_{JL2}, L>. The combination of a past intention achievement, a strategy and an intention that could be realized (i.e. the triplet <J_{ri1}, S, L>) is called a *candidate section* and is a fundamental concept for expressing the operational semantics. At each execution of a section, a new collection of candidate sections (i.e. sections that could be enacted next) has to be computed. Thus, Map enactment engine relies on elements we have introduced above: *Realized Intentions*, *Executed Sections* and dynamically computed *Candidate Sections*. The Map structural meta-model is presented in the upper part of figure 2.

3 BEHAVIOUR META-MODELING

The schema in figure 2 presents an extended Map meta-model with enactment engine architecture (the lower part of the figure). This schema is however static; it does not express how the enactment is dynamically handled by the engine. The purpose of behavior modeling is to capture this dynamicity and to express both the interaction between different elements of the architecture, and the interaction with the tool execution environment (i.e. external application and end users). To specify model behavior, we have introduced in a previous work an event-based notation (Assar et al., 2011).

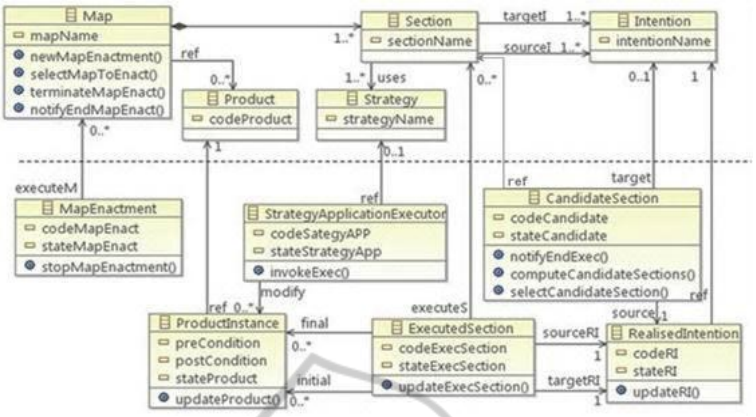

Figure 2: Extended Map meta-model with enactment engine architecture.

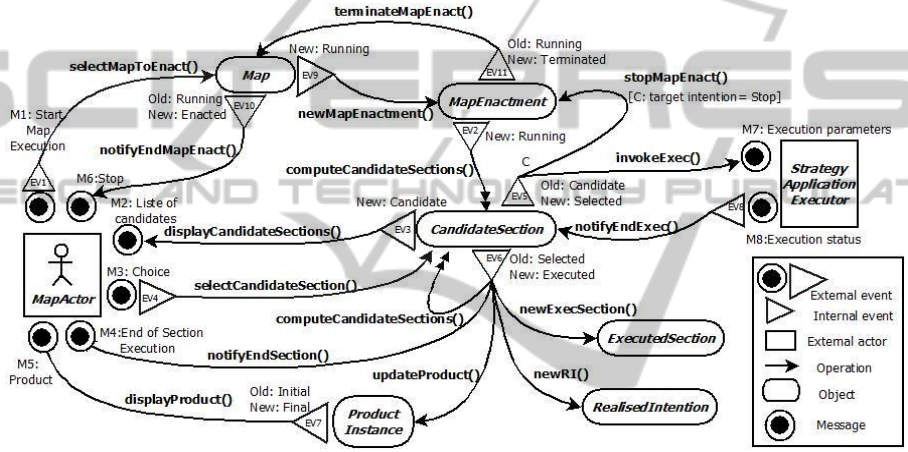

Figure 3: Behavior model for the enactment engine.

The behavior perspective is built upon the concepts of *event*, *trigger* and *operation* (Fig. 3). An *External event* corresponds to the arrival of a message, while an *Internal event* is related to a state change in an object. A *Message* is issued by an agent, which can be a human actor or an application system. The *ascertain* relationship is defined either between an event and a message (for an external event), or between an event and an object in case of an internal event. The *Trigger* relationship relates an event to the triggering of one or several *Operations*. This execution can be conditional; in this case, a specific *Condition* is associated to operation triggering.

The end user (i.e. *MapActor*) selects a map to enact (message *M1* and event *EV1*). This sets the map status to 'Running' (event *EV9*) and creates a new enactment session in the engine (i.e. class *MapEnactment*). Inserting a new instance in *MapEnactment* in status 'Running' (*EV2*) triggers the initial candidate section computation.

When new instances of *CandidateSections* are inserted (*EV3*), they are displayed to the user (message *M2*). The user then selects a candidate section to be executed (message *M3*), and the status for this candidate section changes from 'Candidate' to 'Selected' (*EV5*). This triggers the execution of the strategy which is associated with the selected section. Once the end of the strategy execution is notified (*EV8*) by the external application (message *M8*), the status of selected candidate section changes from 'Selected' to 'Executed' (*EV6*). After updating different data (*ExecutedSection*, *RealizedIntention*, *ProductInstance)*, this triggers the computation of candidate sections (*EV6*) and a new dynamic cycle from events *EV3* and *EV4* is launched.

4 DERIVING AN ENGINE FOR PROCESS ENACTMENT

In order to derive a running tool from these meta-specifications, we are working on transformation rules that target Java platforms. Because of the

dynamic and interactive nature of the behavior model, we rely on the publish/subscribe development patterns. These patterns originate from the field of distributed programming and were initially proposed for designing loosely coupled systems (Eugster et al., 2003). Subscribers can express their interest in an event, and are automatically notified of any event, generated by a publisher, which matches their registered interest. An event is thus asynchronously propagated to all subscribers that are registered to that event. This pattern of asynchronous interaction is being recognized as the paradigm of choice for reactive application development (Hinze et al., 2009).

Three main rules are necessary to transform the behavior schema into a reactive and dynamic running application (Fig.4). The first rule concerns the internal event. It says that each event is transformed into a *Listener* class type, and all operations triggered by this event will be called in the *firePropertyChange* method of this class. For external events, we distinct tow cases. In the case of an actor action, the rule consists in transforming the actor object into a *Publisher* class having methods that allow adding, removing and subscribing listeners. The third rule concerns the invoking of an external actor. In this case, the trigger which displays messages to the end-user is transformed into *Publisher* class type in a similar manner to the second rule.

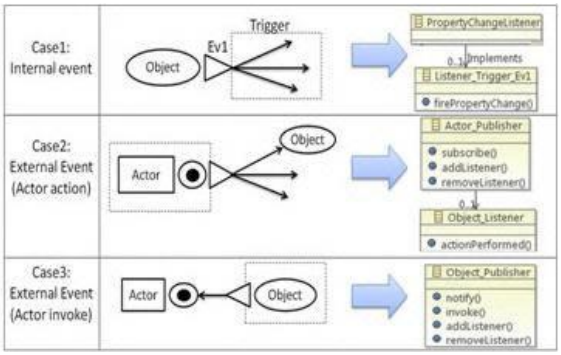

Figure 4: Transformation rules from event model concepts into publish/subscribe patterns.

This work is under progress, and we are actually exploring how to express these rules using some formal notation such as ATL.

5 CONCLUSIONS

We have presented in this paper an approach inspired by MDE for designing and developing software tools. In this approach, meta-models are fundamental artifacts which are used to express both the structure and the operational semantics of target models that will be manipulated by tools. We claim that using adequate transformation rules, a fully running software tool can be obtained. This proposal is ongoing, and we are working on the specification and the implementation of the transformation rules. The contribution of this work is in model engineering and in process model enactment. We seek to rigorously specify the operational semantics of process models and to apply it to a decision oriented model used to describe and to guide engineering process. The proposed approach is promising, it has been partly experimented, it needs however further validation and formalization.

REFERENCES

Assar, S., Mallouli, S. & Souveyet, C., 2011. A behavioral perspective in meta-modeling. *6th Int. Conf. on Soft. and Data Technologies (ICSOFT)*, Sevilla, Spain.

Breton, E. & Bézivin, J., 2001. Towards an understanding of model executability. In *Proc. 2nd Int. Conf. on Formal Ontology in Inf. Syst.* USA: ACM.

Edme, M., 2005. *Proposition pour la modélisation intentionnelle et le guidage de l'usage des systèmes d'information*. PhD thesis, University of Paris 1, France.

Eugster, P.T., et al., 2003. The many faces of publish/ subscribe. *ACM Comp. Surveys*, 35(2), p.114–131.

Hinze, A., Sachs, K. & Buchmann, A., 2009. Event-based applications and enabling technologies. In *Proceedings 3rd ACM Int. Conf. on Distributed Event-Based Systems*. New York, USA: ACM, p. 1–15.

Kermeta, 2012. http://www.kermeta.org.

Olle, T.W. et al., 1991. *Inf. Syst. Methodologies: a Framework for Understanding*, Addison-Wesley.

Rolland, C. et al., 1988. The RUBIS system. In T.W. Olle, A.A. Verrijn-Stuart & L. Bhabuta (eds), *Computerized Assistance During the Information Systems Life Cycle*, North-Holland, p. 193–239.

Rolland, C., Prakash, N. & Benjamen, A., 1999. A Multi-Model View of Process Modelling. *RE*,4,p.169–187.

Souveyet, C. & Tawbi, M., 1998. Process centred approach for developing tool support of situated methods. *DEXA'98*. LNCS. Berlin/Heidelberg:Springer, p.206–215.

Sprinkle, J. et al., 2011. Metamodelling: State of the Art and Research Challenges. In H. Giese et al., éd. *Model-Based Engineering of Embedded Real-Time Systems*. LNCS. Berlin/Heidelberg: Springer, p. 57–76.

Velez, F., 2003. *Proposition d'un environnement logiciel centré processus pour l'ingénierie des systèmes d'information*, PhD thesis, University of Paris 1, France.