Generation of Dynamic Process Models for Multi-metamodel Applications

Fahad R. Golra
IRISA / Université Européenne de Bretagne
Institut Mines-Télécom / Télécom Bretagne
Brest, France
fahad.golra@telecom-bretagne.eu

Fabien Dagnat
IRISA / Université Européenne de Bretagne
Institut Mines-Télécom / Télécom Bretagne
Brest, France
fabien.dagnat@telecom-bretagne.eu

Abstract—Multi-metamodel application development processes have to deal with specific needs including flexible support for structured artifacts like models, multi-layered modeling support and dynamic process updates. In order to deal with the requirements of dynamic process updates and multi-layered modeling support, we propose to model the processes in multiple abstraction levels. The use of conditions on structured artifacts, considered as models, allows enhanced activity interactions, specifically for iterative interactions. This endeavor presented as CAMA Process Modeling Framework (CPMF) counts towards the greater goal of automation of software development processes in the future.

Keywords—Process; Metamodel; Multi-metamodel development; Component based paradigm; Transformation

I. AUTHOR_COPY

Publisher holds the copyright of the original artical. This is the author version of an article published as:


II. INTRODUCTION

The development of Software System is structured in a manner that the time to market and development costs are minimized, without any compromise on the quality of software. In order to achieve this goal, different methodologies are being used, the latest of which is the use of model transformations. Various de facto standards (like UML, XML, SOAP, and so forth) have been adopted by the industry when OMG announced its Model-Driven Architecture initiative. Some even consider MDD to be the first true generational leap in software development since the introduction of the compiler [?].

Modeling was once used to guide the development of software systems, but was not considered as a part of the developed system or the system under development. Advancements in software engineering have changed this point of view. Now, models are themselves considered as part of the system under development [?]. In terms of structure, these models defined by or conforming to their metamodels offer flexibility and variability to software development in two main ways. First, defining a metamodel offers a fair level of conformance for different variations of models. Second, defining multiple metamodels allows the flexibility to exploit different transformations of models in a cost/time efficient manner. Apart from the structure, these models can target different goals, intentions or phases during the development process (documentation, verification, code generation, etc). Each model is considered as a part of development, and can be transformed into other model(s). The process of this transformation may continue till the final source code and the associated artifacts (e.g. documentation and configuration files) for the system under development [?].

Even though, a lot of approaches have been presented in this decade, still a corresponding process modeling support for MDE is missing [?]. [?]. One of the factors restraining process modeling domain from exploiting the benefits of MDE is the use of models as a black-box artifacts, without taking the semantics into account at the meta-level. A process can benefit from the semantics of a model (as input resource) only if it handles models as structured artifacts rather than just black-box artifacts, which allows to access their structure, the related constraints and reason about their mappings to the semantic domain. For example, a process that takes a model containing OCL constraints, as an input artifact, can benefit from these constraints to guide any modification on the model. We also believe that a unique metamodel is insufficient to capture all the semantics of process modeling at different phases of software development. The constructs used at the deployment phase of software development are quite different from the constructs of specification level, so are their relations to the real life concepts, which changes the semantics in varying development phases. For example, the semantics of software components vary between architecture, implementation and deployment phases [?]. A binding in component based paradigm refers to a dependency between components at specification level, whereas it means some invocation at the implementation level, and refers to the network in the deployment phase. It is hard to define process models for a very evolving product line of software, where the semantics of the concepts evolve with time. For these reasons, we
propose a process modeling framework for multi-metamodel application development that can cope with this evolution.

The significant inclination towards Model Driven Engineering (MDE) in the software industry is reshaping the way development process was conceived. Numerous process modeling approaches have been devised in the recent decade to offer features like support for computer aided software engineering (CASE), formal semantics and execution. Still they are not getting the best out of MDE, for not exploiting the use of multiple metamodels. The influence of general process modeling over these domain specific modeling methodologies have lead to a plethora of languages, most of which are based on workflows. We believe that in specifying a process as a collection of activities, the workflows should not be the principle focus. Process modeling domain is dominated by business process models that focus on the workflow and sequencing [7]. Such process models are a good choice to model and compose existing web services [7], [8]. But we believe that they are unsuitable for modeling a wider range of software development processes. For example, BPEL [7] suits quite well to model the process orchestration involved in an online billing system, while BPMN [7] lacks the flexibility to model a higher order transformation process. Reason being, the inputs and outputs of a higher order transformation process are processes (transformations) as well [7]. For these reasons, we argue that the main focus in modeling a process should remain on the semantics of the activities.

One of the important factors restraining software industry to unleash the potential of Model Driven Engineering is the lack of conceptual correspondence between software process modeling and software development paradigm [7]. The specific approaches for software process modeling have been worked upon both by academic research propositions and industrial standards. Still, the visible influence of reliance on workflows and the absence of concepts like typing, execution, implementation, etc., mark the gap between software process modeling and software development paradigm. The conceptual correspondence amongst the two domains, i.e. process modeling and software modeling, can be achieved by promoting the usage of multiple abstraction levels in process modeling, as in software development paradigm. Use of multiple abstraction levels promotes the separation of concerns between implementation level problem definitions and the abstract level constructs. Besides this, such an approach favors process improvement, not only at organizational level but also within a project. This can be achieved if an activity can be replaced or updated dynamically without affecting the activities depending on it. The implementation level processes can be dynamically updated if the mappings towards the abstract level constructs are honored. We propose to tailor software process modeling to mimic component based software engineering in our approach. The influence of component based paradigm guides the development of process components to the use of specified interfaces. As a consequence, the activity interactions within a process are more inclined towards event based dynamic component interactions, rather than workflows. This moves the focus of the process model from data-flow to control-flow. A workflow is based on the deliverance of artifact from one activity to another, whereas the component based mechanisms focus on the control-flow of interactions based on dependencies, which guides the data-flow as well.

Figure 1. Process Metamodels for Multi-metamodel Development

To summarize and explain premises for this article, we present a little scenario of a higher order transformation process [7]. The nature of this process demands the modeling technique to model the process in both phases (transformation definition vs transformation execution). It expects to reuse certain process definitions for different variations of transformations and support domain specific modeling specifically tailored for processes. It can also benefit from the use of the structure and semantics of input artifacts (transformations). It would be better for the process model to deal with multiple hierarchical layers (working with transformation definitions, transformations, metamodels and models at varying abstraction levels). All these premises are asserted by referencing this scenario while explaining the framework and the relevant metamodel packages for our approach. A general overview of modeling this scenario with our approach would shift the focus from the data-flow between the activities to the control-flow. If this transformation process is modeled using some workflow, the focus remains on the flow of artifacts from one activity to another, which guides the control-flow. As in this case, a sub-activity patternMatching passes the list of matching patterns to another sub-activity ruleGeneration. But when modeled using CPMF, the data dependency between the both is modeled at a higher abstraction level, whereas the control-flow between the activities is modeled at a lower abstraction level, using events. CPMF framework would also model this scenario using different metamodels at specification level and implementation level. This promotes flexibility and reusability, in case you want to reuse the specification model instead of the instance model of the scenario.
The novelties of our approach are: a) capturing the semantics of models at different phases of development, b) using structured artifacts to access their semantics (if needed), c) enhancing the dynamism of the process model by allowing on the fly manipulations, d) promoting re-usability for activities by allowing multiple activity implementations, e) decoupling the activities by using event based mechanisms and f) exploiting multi-level modeling by introducing (abstract and concrete) abstraction layers.

This paper is structured as follows. Section ?? describes the holistic view of the approach used in CAMA Process Modeling Framework. Section ?? describes the Implementation Process Metamodel to explain the specific syntax and (informal) semantics of the metamodel used for processes. Section ?? presents a case study applied on a classical process modeling problem. Section ?? compares our approach with other recent endeavors in the field of process modeling. Finally Section ?? concludes and outlines the future perspectives.

III. MULTI-METAMODEL PROCESSES

Software development through the evolution of models from requirements till deployment passing through a series of transformations is the hallmark of MDE. These transformations are responsible for creating, modifying, merging or splitting models, as the software development project advances. We argue that capturing the semantics of the processes at different development phases is difficult using a unique process model. For this reason, our approach presents two metamodels i.e. Process Specification Metamodel and Process Implementation Metamodel, illustrated in Figure ??.

The Process Specification Models are developed in conformance to the Process Specification Metamodel. A Process Specification Model is used to document the process best practices, where it is not overloaded with the implementation level information. It is not specific to any organization or project. This added level of abstraction in terms of specification promotes re-usability of the process models. Process standards and best practices are documented in a reusable manner, where they can further be applied to any specific project or organization in multiple specific manners. The Process Implementation Model conforms to the Process Implementation Metamodel. A Process Implementation Model documents the specific project details, which are incorporated in the model by adding the implementation details of the process model. The Process Implementation Metamodel is semantically more rich, so as to express all the fine details of the process model in terms of implementation.

Let us take an example of an organization that develops processes at different development phases to promote automation within the process. Thus our approach focuses on defining a dedicated process model for each phase, and to use model transformations in between to promote automation within the process.

IV. METAMODEL (PIMM)

We are presenting the Process Implementation Metamodel in this article for the reasons that this metamodel is semantically richer than the Process Specification Metamodel.
Process Specification Metamodel presents the specification level details and avoids to pollute the model with implementation level details. We have already presented the Process Specification model under the name of general process metamodel in another paper [2]. This paper focuses on presenting the framework, with the addition of the Process Implementation metamodel, where the focus lies on the separation of concerns, the usage of event based mechanisms and the dynamism introduced through multi-level modeling. Process Implementation Metamodel defines the overall structure of the implementation level processes. The metamodel is built from three packages: Core package, Activity Implementation package and the Contract package. The two later packages are merged in the former to get the complete metamodel, which is presented using UML syntax.

The Core package of Process Implementation Metamodel defines the five core entities of the process model, as illustrated in Figure ???. A process in our framework is a collection of activities, that show dependencies between them. Each of these activities have their contracts. These contracts allow these activities to interact with other activities (within their context or through a containment relationship). These activities are not sequenced in terms of a workflow. On the contrary, their contracts define if they depend on some other activity for their execution. These dependencies between the contracts are not shown here in this package but will be illustrated in the contract package. Thus, instead of focusing on a proactive control for the process, we stress to focus on the completeness of definition for each individual activity. An activity being the basic building block of the process model expresses its interactive requirements for operation, using these contracts. A process having both activities and their associated dependencies represents an architecture using activities as basic entities and dependencies to define the flow between them. The activities only express their dependencies, and thus the absence of 'hard coded' control-flow introduces a fair amount of dynamism in terms of activity sequencing. This dynamic sequence of flow equips the process model with a reactive control that has the capability of restructuring the control-flow for the activities at runtime.

For example, two independent sub-activities planned to be performed one after the other, may be performed in parallel, in case the project gets late.

The process model is presented as a hierarchical structure, where each activity can contain sub-activities. The concepts of sub-activity and super-activity are not based on generalization, rather we are taking them in terms of containment. An activity in our process model can have multiple super-activities, which means that it is contained by multiple activities. This introduces the concept of activity sharing in our process model. Process Specification Metamodel specializes the activities as primitive or composite, where a primitive activity does not contain any further activities, whereas a composite activity contains sub-activities. Inspirations from the component based paradigm have led us to define an activity as a black box component, where the only interface of an activity to its context or content (in case of composite activity) is through the defined contracts. All the interactions between these activities can only be handled through contracts. This gives a nature of pipe & filter architecture to the process, where the activities serve as filters and the dependencies serve as pipes. In order to add the mechanisms to express the responsibility assignments for each activity, a role entity is used. This entity defines the role that is responsible for performing the activity.

The second package in this metamodel is the Activity Implementation package, shown in Figure ???. This package defines the content of an activity, in terms of the operating description. A DescribableElement in this metamodel is an abstraction for all elements that have a name and a description. We are using three describable elements in this metamodel, as Process, ActivityDefinition and ActivityImplementation. Considering the importance of separation of concerns, SPEM2.0 separated the usage of activities from its contents. Though, SPEM2.0 uses this separation to add variability to process modeling, we have taken it ahead by the inspirations from object-oriented paradigm. We have used the strength of typing concepts, so as to add both variability and conformance to our activity structure. An ActivityDefinition behaves like an activity type (more of
a conformance relationship between two entities.

As the activity implementation package of Figure ?? is a minimal metamodel, it does not show the further specialization of activity implementation in terms of primitive and composite activities. In case of a primitive activity, the activity implementation defines the procedure for its realization apart from the specified concrete contracts. In case of a composite activity, it encapsulates a process, defined in terms of internal activities and their dependencies. All the interactions to/from the context of a composite activity by/to its content, have to pass through its contracts.

The responsibility assignment for the process is a very important aspect for process modeling. The activity implementation package defines this notion using three abstraction levels. Each activity definition is played by a responsibility, whereas each activity implementation is played by a role. A responsibility of an activity definition specifies the level of authority for carrying out a task, for example approver or accountable. Whereas a role for the activity implementation specifies the precise role that enjoys the level of responsibility it refers. Thus a project manager can be the approving authority for an activity, where the software engineer may be responsible for performing the same activity. A team or an actor realizes the role for an activity implementation, where a team has multiple actors as participants.

The third package of the Process Implementation Metamodel is the contract package, presented in Figure ???. All the interactions to/from activity definitions or activity implementations are carried out through their specified contracts. These contracts can either be internal or external to the activity definition or activity implementation, which is defined through the position of the contract. The direction of the contract defines whether it is used for inward communication (required) or outward communication (provided). It should be clear that the interaction from sub-activities to a super activity is also inward for the super activity. In case of

Figure 4. The Contract Package
composite activities, for every external contract, there is a corresponding internal contract of the opposite direction. Thus for a provided external contract, there exists a required internal contract and vice versa (though this point in not depicted in this minimal metamodel).

An activity, being the basic unit of processing, takes work products as inputs, modifies or works on them and produces them as outputs. The input and output work products of an activity are called artifacts and they specify the data-flow within the process. An abstract contract of an activity definition owns artifacts, whereas a concrete contract owns events. This separation of contractual resources, allows to separate the data-flow of the activities from their control-flow. Dealing with the data-flow at an abstract level (apart from the control-flow) allows the data-flow mechanisms to benefit from effective means like data repositories and configuration management. Whereas the control-flow within the activities through the use of events in concrete contracts can be effectively managed by an underlying event management system. One of the major reasons to choose event based mechanisms is to allow the decoupling of the activities for the control-flow. By decoupling these activities using events, the focus is brought back to the completeness of the definition of each individual activity. All the events owned by the concrete contract map to the artifacts of the abstract contract. Thus, having both the levels separated, a link is kept between the control-flow and the data-flow, so that the control-flow should be able to guide the data-flow between the activities, as and when necessary.

One of the key features of our framework in terms of artifacts is the exploitation of structure. Artifacts are not taken as black box entities and their structure is kept comprehensible to the activity. This is enforced by the fact that each artifact conforms to a metamodel. This allows an activity to take an input artifact and even dynamically reconfigure itself (if needed) through the use of that artifact. The syntax and semantics encoded within the artifact is explicitly made accessible to the process that can exploit this information. This further allows us to define semi-automatic processes in our framework, for example model transformations. Again, the model shown in Figure ?? being a minimal metamodel, does not show the use of properties on the contracts. Use of these properties (specially the pre-conditions) on the structured artifacts helps in managing semi-automatic processes with model manipulations. In case of our scenario, a transformation takes a model as an input, and has the capability to verify its conformance to the specified metamodel. Again, the process modeling approach does not accept the artifact as a structured artifact, it can not notify the preceding activity about any problems that may be associated with the input. The concept of using a structured artifact may not be of prime focus for manual activities, but is of high value for modeling the automatic and semi-automatic activities.
Activities specifying their interaction contracts express artifacts at abstract level and events at concrete level. The artifacts and events are the contract contents for these activities. Dependencies between the activities are defined in terms of content mappings. A content mapping relates the contract content of a required contract to that of a provided contract. Each dependency defines the content mapping between the contracts of corresponding activities. These dependencies are used to control processes, and help create a dynamic control/data-flow between the activities.

One of the classic problems in software process modeling was published in the 6th International Software Process Workshop as a benchmark for the purpose of comparing various approaches in the domain [7]. We have also chosen this standard benchmark problem to facilitate the understanding and to present the diverse aspects of our approach. For reasons of brevity, we are only explaining the implementation of the problem, and the readers are referred to the original problem for a detailed understanding. This problem covers the ‘software change process’ portion of the software development cycle, thus focuses on a single cycle of designing, coding, testing and management of an already developed code unit of a software. This software change process may occur at any later stage of the development or even at the maintenance stage. The problem has been formalized in a manner that it can express the real world problems, encountered in software development.

ISPW-6 benchmark problem has been modeled in CAMA process modeling language as a textual model. We are presenting a graphical model in Figure ??, which is only intended for explanation purposes. It should be noted that our approach is currently not providing any graphical modeling language, and its entire scope remains on the semantic representation of process models. This problem presents a single activity ‘Develop Change and Test Unit’ that contains all other activities for carrying out this process. Process Specification metamodel takes the problem as a single process named SoftwareChange, and places this activity in it, as illustrated in Process Block ??.

The DevelopChangeAndTestUnit activity in turn expresses its (internal and external) contracts, in order to interact with other activities, as shown in Process Block ?? These contracts are specified in terms of artifacts, because at this level of abstraction, they are abstract contracts. This guides the data-flow of the activity to/from other activities. This activity expresses its internal contracts through the artifacts like Requirement Change document (that it provides to its sub-activities) and the Approved Modified Design (that it requires from its sub-activities).

Same way, it expresses its external contracts through artifacts like Requirements Change document and Project Plan that it requires from its context. It should be noted that the Requirements Change document is in fact handed over to the sub-activities but passing through the external and then internal contracts of this activity.

This activity is a composite activity and it contains all other activities of this example problem, but still it does not express the details. The reason that this activity does not show any of the contained activities is that these details are expressed at the activity implementation level, rather than the activity definition level. The activity also expresses the responsibility assignment for this activity, which is defined as responsible and authority. The responsible of this activity has the responsibility of performing this activity, whereas the authority of this activity has the responsibility to allow its processing and to terminate it, when necessary.

Process Implementation metamodel specifies the contained process by DevelopChangeAndTestUnit activity. As discussed in the activity implementation package, a composite activity contains a sub-process, which in turn contains other activities and dependencies between them. Thus at the implementation stage of development, this activity expresses its contained process, DevelopChangeTest. This process further contains eight activities and the dependencies between them. The dependencies are expressed in terms of one activity being dependent on another activity, where the precise contracts of the activities are expressed through content mappings, as illustrated in Process Block ??.

The dependencies shown in the process are the direct dependencies (contextual) of the activities within a process.
However, the dependencies of an activity to its super-activity or its sub-activity are expressed within the activity implementations, as delegate dependencies (contentual). One other important aspect to be noted here is that the dependencies are expressed between the events of the activities. As defined in the activity implementation package, every event at concrete contract maps to an artifact at the abstract contract. Thus when dependencies are dealt with in the lower level of abstraction in terms of events, both the control-flow of the activities (directly) and the data-flow between them (indirectly), can be manipulated. The concrete level of abstraction also allows the dependencies between roles and contracts. This allows an activity to interact with the specified roles of any other activity. ScheduleAndAssignTasks activity carries out the task assignment using these dependencies.

Process Block ?? illustrates the activity implementation for the DevelopChangeAndTestUnit_def activity definition. This is one of the activity implementations and a model may contain many alternative activity implementations for a single activity definition. This activity implementation specifies the process contained by the activity. It also defines the internal and external contracts of the activity. The internal contracts of the activity are defined in terms of events like RequirementsChangeEvent_P, which is a provided event that refers to the Requirements Change document specified in the upper level of abstraction (the abstract contract). Thus all the actions concerning this event, can indirectly be applied to the artifact at the upper level of abstraction. Same ways, the external contracts of the activity are also defined in terms of events like RequirementsChangeEvent_R, which is a required event that refers to the Requirements Change document in the abstract contract.

The delegation dependencies of this activity implementation are expressed on the super-activities or sub-activities (no sub-activities in this case). These delegate dependencies are also specified in terms of concrete contracts, which map to abstract contracts. Thus using the dependencies specified on the events, both the control-flow and the data-flow can be manipulated. Finally, the roles for the activity implementation show the performers of the activity, which refer to the responsibility in the upper abstraction level. The Configuration Control Board role for this activity has the authoritative responsibility for this activity implementation, whereas the project team is responsible for carrying it out.

VI. RELATED WORK

Various approaches have been proposed to model and execute processes. The scope of each of these languages have been overlapping in the sense that they all are targeted to model processes, but they have not actually addressed the same issues as in the core of CPMF. Our focus in this work is not on the graphical modeling of the processes, as done by most of the well established process languages like Little-JIL [?], SPEM2.0 [?] and many other approaches based on UML activity diagrams [?], [?]. Little-JIL defines the operational semantics of the language and puts
them all in a graphical syntax, which complicates the end visual model. We believe that in case of simple and lesser complexities, graphical notation is a more viable choice for modeling, but as the concepts grow towards complexity and a complete definition of all the abstractions is needed, a textual model is a better choice. We have restrained ourselves to the textual model, where the focus has remained on the operational semantics. Thus the definition of translation to a graphical notation is not formally carried out yet. But we are looking forward for the possibility of reusing a well accepted graphical notation in process modeling for our own process modeling language. A mapping of such graphical notation would allow us to visualize the textual models.

The concept of using multiple abstraction layers in CPMF, has also been used by Chou's method [2], but the intent has been different. Chou uses the higher level for UML based diagrams and then translates them to a lower level using a process language. This usage of multiple layers translates the graphical notation to the textual notation, where the semantics remain the same at both levels. We on the other hand, use the two levels to separate the concerns of abstraction. The higher level is an abstract level that defines the types of the process structures, whereas the lower level is the concrete level which is used to define multiple implementations based on the definitions of the types at higher level. This separation of concerns is also tackled by SPEM2 [2], by separating process structure from method content, but they rely solely on defining different variations for task usage based on the task definition. It does not present any typing/conformance relationships, where activities can be implemented in different ways based on the same type. This separation in SPEM2.0 works fine for the roles, work products and tasks, but it places activity as one complete entity in the process structure. So no separation is done precisely for activities and processes, and the focus has remained on the separation of their resources in terms of content and usage. We on the contrary have focused on the activities, which exploit the implementation/conformance relationship to benefit from the usage of multiple abstractions.

Processes in CPMF are inspired from component based paradigm, which has been targeted by MODAL [2] and SPEM2 as well. Here again, SPEM is using the concept of process component for specifying the work products of the components in terms of ports, whereas we are more close to MODAL that uses ports for providing services. We use the concept of contracts to specify the artifacts at the abstract level, however these contracts are mapped from the contracts at concrete level which specify the services in terms of events. So a contract is technically used for data-flow and control-flow at the same time, but their specializations separate the two usages, so as to deal with them individually in an effective manner. The event based mechanisms used in CPMF, have also been used by OPSS [2], but they have used it to model processes for agent based systems. Our scope has been to unleash the event based mechanisms to add dynamism and reactivity to process modeling.

Some of the recent approaches like UML4SPM [2] and xSPEM [2] have targeted towards the executability of process models. They have implemented this by mapping process models to XPDL or WSDL for executing processes as services. xSPEM has also offered model validation by translating the properties on SPEM into LTL properties on the corresponding Petri Net models. BPMN [2] and BPEL [2] as a couple provide a good alternative to our approach that scopes the designing and the execution phases of software development. But the lack of conceptual correspondence between the both makes it hard to synchronize the models, specially when the software development lifecycle models are getting more iterative. We have the possibility of translating a BPMN model to a CPMF model, however a well defined CPMF process model can not be transformed to BPMN or any other approach, without the loss of information. Our novelty in CAMA Process Modeling framework is to capture the semantics of process at different stages of development. Such an approach does not only support model driven engineering for the application development, but also uses it to generate the dynamic process models. The defined metamodels for the CAMA Process Modeling framework have been formally developed as Ecore models. We have defined the grammar for our process modeling language, that conforms to these metamodels. In order to provide the tool support for this framework, we have already developed a textual editor based on Xtext. We are heading on to present our own Process Instantiation Metamodel for CPMF, which would also be formalized by specifying the behavior of each individual activity and the interactions between them.

VII. Conclusion

This research project promises a process modeling approach that models activities as components, having their defined contracts. The overall hierarchical structure of the process metamodel allows to model processes at different abstraction levels. Moreover, CAMA Process Modeling framework is aimed to be defined using three metamodels i.e. Process Specification metamodel, Process Implementation Metamodel and Process Instantiation metamodel, which helps to capture the semantics of processes at all development stages. By accepting models as required and provided artifacts and extending support for model transformations, processes are adapted with the capabilities to support Model Driven Engineering. The instantiation semantics for these processes is being sought, which would allow us to have an executable software development process modeling approach. Furthermore, we are currently working to define a formal semantics of these processes. Such formal semantics would allow us to verify the properties of the processes all the way starting from the requirements phase till the deployment. We are also looking forward to model complex
requirements engineering processes from the industry, that
would allow us to quantify the effectiveness of our approach.
Our basic implementation of CPMF would provide us with
adequate tools to model the processes, where we have the
possibility to reason about their semantics for verification.
For a complete tooling support, we have chosen to extend
Openflexo [?]. Openflexo is an open source process model-
ing tool using BPMN that allows the definition of processes
and assignment of resources and roles in a well formed
graphical user interface. We want to extend it to support
the CPMF modeling language to generate multiple process
models. An interpreter of an instance model would also be
added to Openflexo to support process execution.