M2Flex: a process metamodel for flexibility at runtime

Eric Céret[1] Sophie Dupuy-Chessa[2] Gaëlle Calvary[1]
Grenoble Institute of Technology[1], Pierre Mendès France University[2]
Grenoble Informatics Laboratory
41 rue des Mathématiques
38041 Grenoble Cedex 9, France
FirstName.Name@imag.fr

***Abstract*— Design and development methods do not meet designers' and developers' needs. They are difficult to learn and to use; they are complex, sequential and rigid and thus far from being adapted, reliable and efficient.**

This paper presents M2Flex, a process metamodel for highly supporting flexibility. M2Flex is based on a recent definition of flexibility along four dimensions: (1) variability, the existence of equivalent choices, (2) granularability, the existence of different levels of details, (3) completeness, the possibility of defining optional components and pre-defined reusable results, and (4) distensibility, the capacity of the resulting process model to be extended or reduced at runtime.

This paper shows how M2Flex is original by the flexibility it offers to designers and developers at runtime.

***Keywords*—Process Models, Flexibility, Design Methods**

I. INTRODUCTION

Designers and developers are poorly satisfied by methods [1, 2, 3]. They report that methods (1) do not address various kinds of projects and customers' constraints, (2) are difficult to learn and to use, (3) impose complex, linear and rigid processes that are not described in adapted languages. The authors of the studies conclude that the process models of the methods are not flexible or adaptable enough. According to [4, 5], the process model is part of a method, with the product model and a collection of tools. It focuses on a facet of the design and development process - e.g. the tasks to complete, the products to build or the decisions to make - to describe the activities to be realized.

In this paper, we investigate the flexibility of process models. Many researches [6, 7, 8, 9, 10] have been driven to evaluate it. In particular, Harmsen, Brinkkemper and Oei [8] defined a one-dimension classification (Figure 1) for measuring it, ranging from rigid models to the modular construction of process models.

In [11], we proposed a taxonomy for evaluating and comparing process models, based on the study of 49 of them and on several previous works. This taxonomy makes it possible to classify process models with any orientation (activity, product, goal,...) and offers a new definition of flexibility, based on four dimensions: variability, distensibility, completeness and granularability.

Variability is the possibility offered by a process model to designers of making choices in a set of equivalent variants. For instance, the goal "write the use case" can be achieved by several variants, which can be single activities like the creation of a UML use case or the drafting of a free language text (with Microsoft® Word, OpenOffice™, Google Document© or on paper). The variants can also be sequences of activities, like filming a user who expresses his needs and then transcribing the video. Variability can also concern other elements of the process model, such as the choice between equivalent artifacts (e.g. documents with different structures but the same information).

Granularability is the ability of a process model to support elements with different granularities, e.g. with different languages and/or quantities of details. For instance, if the process model includes an activity for defining the structure of a database, it can suggest a goal "create the database". An expert database designer will not need more information. It can also iteratively define activities with more detailed steps, offering a step-by-step approach for novice designers.

Completeness is the possibility of fulfilling or not the proposed process, some activities and/or artifacts are then optional or can be replaced by a predefined result or product. For instance, in a User Interaction (UI) design, the activity "define the platforms model" can be avoided; in this case, the UI is then designed for an implicit platform, or, if this model is needed because several platforms have to be addressed, it can be replaced by "default" models that the designer picks up in a repository proposed by the process model.

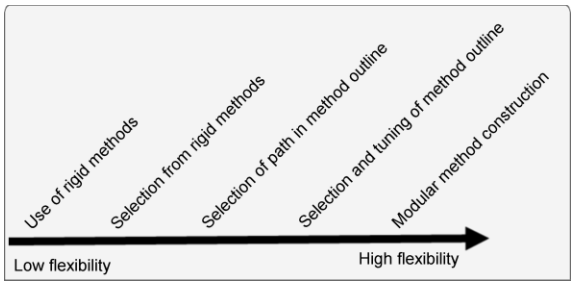

Fig.1 - Flexibility levels in process models according to Harmsen *et al.*

Distensibility is the ability of a process model to be extended or reduced, i.e. to accept that proposed elements (such as activities, roles and artifacts) can be avoided or that unexpected elements can be added. The question here is not if someone did extend a process model: this has been achieved several times, for instance in [12], where the Case-Based Reasoning Process is presented as an extension of intentional reminding [13], or in [14] where Scrum [15] is extended to manage several teams working on the same project. The issue is here the definition of mechanisms for distending the process model during its enactment.

Many examples in the taxonomy show that process models implement only partial flexibility. For instance, Rapid Application Development (RAD) [16] is classified as offering some well-defined variants but no possibility of extension, supporting partial incompleteness and including four levels of granularity.

Based on our taxonomy, we propose a metamodel and an editor for creating flexible process models that suit the needs of designers and developers at enactment-time, i.e. when they are selecting and then realizing their design activities and producing the artifacts. The novelty does not only come from the flexibility dimensions covered, but also from the tool that enables the process models adaptation during enactment.

We present this new process metamodel in the second section and analyze its flexibility in the third section. In the following section, we present a process model compliant with our metamodel and then describe the existing tools supporting our approach. In the sixth section, we compare our proposition to existing works, and conclude with perspectives.

II. PROCESS METAMODEL

The metamodel description is divided into two main parts: a global explanation of the metamodel first, followed by details about the packages.

A. *Global overview of M2Flex*

Figure 2 presents the packages of our process metamodel, M2Flex.

A process model is here considered to be realized in purpose, so it is composed of some main goals, as it is done in many goal-oriented process models such as KAOS [17], I* [18] or MAP [19]. As it is possible to achieve any of these goals in many ways, the various possibilities are represented as **strategies**. A strategy can be associated to a **condition**: for instance, adopting a User Centered approach requires that users are available.

Strategies are concretized into **activities**, representing the operational tasks to be realized. In order to represent various amounts of details, activities can be elementary or composite - and then refined into other simpler activities.

Some activities can be carried out in parallel, like the coding of a functionality and the creation of its unit tests, whilst others need to be strictly sequenced: delivering a new release to the customer can only be done after implementing it. M2Flex includes a mechanism for expressing how activities may be combined - this is detailed in the section about operators.

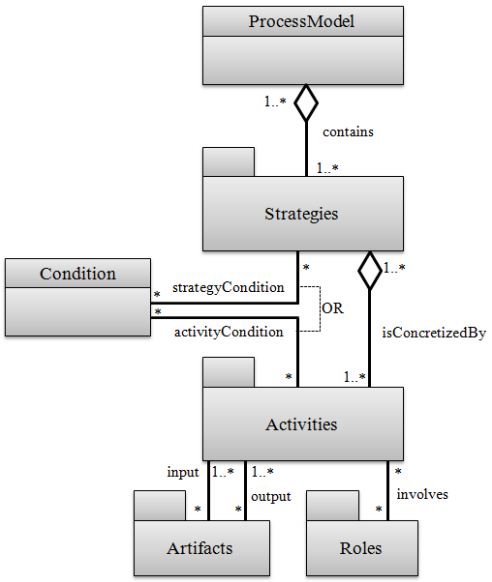

Fig.2 - Process metamodel overview

However, representing how activities can be combined is not sufficient to express all the conditions that have to be satisfied before an activity can be started. Indeed, activities often require that another task has been completed and has produced some results. This is why activities are associated twice to **artifacts**, once as inputs and once as outputs. The strategies are thereby associated to the artifacts *via* the activities. Therefore, the strategies implicitly have inputs and outputs. At enactment-time, this makes it possible to graphically represent the process in two ways: either showing first the activities (focusing on the "how", the artifacts being considered as a result), or showing first the artifacts related to the strategies (focusing on the "what", the activities being then presented in a second time as a guide for producing the chosen artifacts). The developer is thus able, at enactment-time, to choose the view he is the more comfortable with.

Activities are also related to **roles**, in order to express that some competencies might be needed to complete the task.

However, there can also be conditions that depend on artifacts, roles or operators. For instance, in RAD [16], the activity "*consider attendance* [of individuals in other corporations]" is realized only if "*the system serves* [such individuals]". Such a condition can not be expressed by an association to artifacts. This implies that activities also have to be associated to conditions.

Hereafter, we detail all the packages that briefly are presented in this section. In all diagrams, the attributes whose name is followed by ² (e.g. status²) are "simple fields" of "deep instantiation" [20]: when reifying the metamodel, they are instantiated into identical attributes at model level (and, as usual, into values at object level). We use this mechanism to impose, at the metamodel level, attributes that are needed at model level. Moreover, the classes that are directly composing the described package are drawn in grey, whilst the elements from other packages are drawn in white. Furthermore, as we

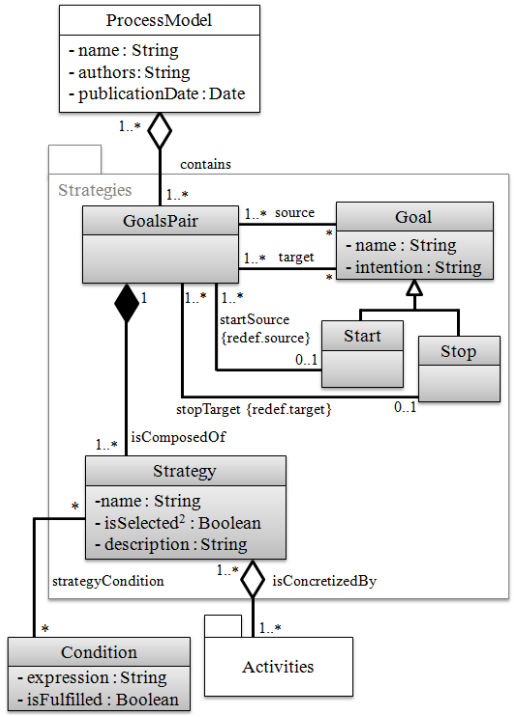

Fig.3 - Package Strategies

focus here on flexibility, all attributes are not extensively presented.

B. *Strategies*

Figure 3 presents the details of the Strategies package. A process model has a name, authors and a publication date. It is an aggregation of elements of the Strategy package. A strategy can be reused into several process models.

We want to express that various strategies can lead from one main stage of the process model to another. Inspired by the work done for defining the Map metamodel [19], we model this package with goals and strategies.

The *GoalsPair*s represent couples of goals, each of them is associated to one source and one target *Goal*s. A goal represents an important objective of a process model. For instance, RAD process model [16] defines four stages: (1) requirements planning, (2) user design, (3) construction and (4) cutover. Representing RAD in our metamodel would require to convert these stages into goals, like "plan the requirements" and "design the system". A goal has a name and an intention, which is a description of its purpose.

The *Start* goal represents the starting point of the process, where no activity has been started, and the *Stop* goal represents its ending point, where all needed activities have been achieved. At least one pair of goals must have the *Start* goal as a source and at least one goals pair has the *Stop* goal as a target.

A *GoalPair* is composed of several equivalent *Strategies*, which are different ways of achieving a goal. For instance, requirement analysis in the V Model [21], modeled here as the goal "describe the requirements", could be reached using a User Centered approach [22] or the Map approach [19].

A *Strategy* is defined by a name and a description. At enactment-time, it can be selected or not by the designers: this is modeled by the isSelected attribute with deep instantiation. A *Strategy* can be associated to some *Condition*s that represent the constraints that have to be fulfilled before the strategy starts. For instance, in a requirements analysis, a user centered strategy [22] requires the agreement of the customer and the availability of some end users.

C. *Conditions*

The *Condition* class expresses the constraints which strategies and activities are subjected to. It has two main attributes: its expression, which can be evaluated as true or false, and an attribute isFulfilled with deep instantiation that is valued during process enactment, indicating whether the condition is satisfied or not. A condition can be associated either to (at least) an element of the Strategies package, or to (at least) an element of the Activities package. The same condition cannot be associated to a strategy and an activity: a strategy being an aggregation of activities, it is hard to imagine that the same condition could apply to both these elements. However, a condition can be associated to several strategies or to several activities. Indeed, when many strategies are equivalent, many of them may depend on the same constraint. Similarly, activities from equivalent strategies may have the same constraints.

D. *Activities*

We aim to represent here the tasks (that we name activities) to be realized during the design and development process. Elementary activities only could not represent all the existing possibilities. For instance, it would not be possible to represent the Scrum Sprint [15], with its iterative sequence of activities that includes activities like "update product backlog", "sprint planning meeting", or "product increment". Thus, we represent the activities using a composite pattern made of *Activities*, *ElementaryActivities* and *ComplexActivities*.

As shown on figure 4, a *Strategy* is concretized by an *Activity*, which is composed of *ElementaryActivities* and *ComplexActivities*. The composition of elementary and complex activities as well as their order are ensured thanks to artifacts and operators in the *ComplexActivity* class. Explanations about the operators refer to the artifacts and the roles and are thus detailed below. An activity can be iterative. For instance, as already mentioned, the Scrum Sprint would be represented as an iterative *Activity*. It also can be incremental, meaning that the resulting artifacts are built incrementally.

An elementary activity has a name, a type[1] (requirement analysis, coding,...) and an allocation (human, interactive or system task). The isOptional attribute is computed at process model enactment-time. It means that the activity can be **not** executed, i.e. that there is a path in the process model that does **not** include this activity. We will see later that this is useful for verifying the validity of the process model.

[1] Enumerations are not displayed on the figure here for a reason of space. They are described in the text.

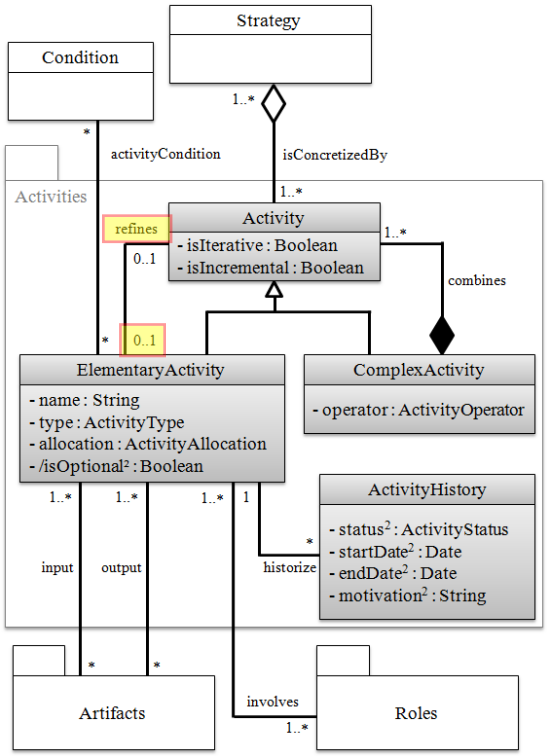

Fig.4 - Package Activities

The elementary activity is associated to *ActivityHistory* so that to preserve information about its evolutions. Therefore, four attributes are valued at enactment-time, the status of the activity (selected, rejected, running on,...), the start and end date of this status and the motivation of the evolution. Thus, all decisions made during the process can be tracked.

An elementary activity is also associated to artifacts and roles: this is described in the next sections.

An elementary activity can be detailed thanks to its *refines* relation to an activity. This represents the possibility for an activity to be detailed at a lower level.

E. Roles and Artifacts

As shown on figure 5, an activity is associated to at least one *Role*, representing the human agents and/or the tools required to achieve the task. Several attributes and specialized classes (human, tool,...) would be needed to represent all required details, but they are not represented here, because they are not relevant for discussing the process model flexibility. The kinds of roles that can be associated to an activity depend on the allocation of this activity. For instance, tools can only be associated to interactive or system tasks.

Activities may also be associated to Artifacts: they can input artifacts and output new or modified artifacts. It is not required that an activity is associated to any artifact. For instance, "notify the customer that a new release is available" has no output and the first activity of the process has no input. When activities input and output artifacts, these artifacts are associated with a specific status. For instance, the activity "validate the requirements with the users" inputs a requirement record with status "draft" and changes this status either to

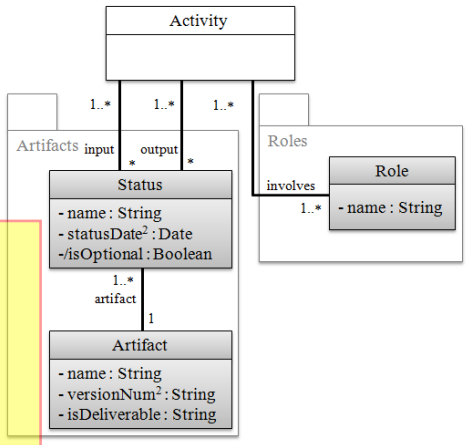

Fig.5 - Roles and Artifacts

"validated" or to "rejected". This is why activities are in fact associated with a status, which is in turn associated with an artifact.

An artifact status has a name, a date that is valued at runtime and a calculated attribute *isOptional*: the status of an artifact is said optional when there is a selectable path in the process model in which there is no activity producing this status. An artifact has a name and an attribute indicating if the results of the activity are deliverable. It also has a version number.

F. Operators

The *ComplexActivity* class offers operators between activities and elementary activities. These operators are based on the operators used in task modeling [23]. All task modeling operators are not necessary here. For instance, the distinction between 'enabling' ("*one task enables a second one when it terminates*") and 'Enabling with Information Passing' (when it terminates, one task enables a second one and provides some information to it) is not required, because it is implemented thanks to the inputs and outputs of activities. After analyzing which operators are relevant here, we defined a set of 5 n-ary operators. In the following description, the word element refers to an activity or an elementary activity.

- **Sequential enabling**: after a source element is achieved, targeted elements can start. This operator is not useful when the dependency between two activities can be expressed using the artifacts: if activity B depends on one artifact produced by activity A, the dependency is already expressed when associating these activities with the artifacts. However, the operator is useful when B does not depend on an artifact produced by A but there is a need that A is achieved before B starts. For instance, coding does not require that the step of writing the function unit tests is completed, but the process may intend that the developer ends it before beginning to code. Without this operator, it would not be possible to express this constraint.
- **Parallel**: the targeted elements can be realized in parallel. For instance, a process model can specify - in opposition with the example given in the section about sequential enabling - that coding and writing unit tests can be realized

at the same time by different roles. The two activities would then be combined by a parallel operator, each of them being associated to a specific role.

- **Choice**: the designer or the developer can choose between some equivalent activities, i.e. activities that produce similar results or outputs. This can also make an activity optional. For instance, the process proposed with the Oxygen code generator [24], proposes the creation of several components in which (a) Data Access Functions, (b) the optional "Data Access Interface" and (c) the Data Transfer Objects. This can be represented by a choice between (b) and (c) and sequential enabling from (b) to (c): one (a) is achieved, the developer can realize (b) and then (c) or directly (c). (b) is in this case optional.
- **Interleaving**: with this operator, targeted elements are executed in parallel by a unique agent/role, who can switch from an activity to another when he wants. For instance, an activity such as "record anomalies" can be done by the same person and at the same time that "carry out unit tests". This operator is a variant of the parallel operator and is above all useful when enacting the process model, because it is needed by model-driven user interface generation [23]. For instance, the activities of an interleaving operator could be converted into different workspaces on the same window, whilst parallel activities would be transformed into independent windows accessible with a menu.
- **Disabling**: the targeted elements are disabled when the source element is achieved. For instance, when the final acceptance is signed by the customer, it disables the iterative implementation of the functionalities.

G. Temporal organization

Artifacts play a key role in the temporal organization of activities at runtime: if an activity requires an artifact with a specific status, then the process model must contain another activity producing this artifact with this status. This constraint is used to validate the structure of the model at design time. This constraint can be expressed as (1):

$$\forall a1 \in Activity, \forall s \in Status, \\ s \in a1.input \Rightarrow \exists a2 \in Activity, s \in a2.output \quad (1)$$

Another constraint (2) is that when an activity requires an optional artifact status, then this activity must be optional itself, because there is no guaranty that the needed artifact will be produced. At runtime, it is used to unselect automatically activities depending on an optional (and unchosen) status.

$$\forall a \in Activity, \forall s \in Status, \\ (s \in a.input \wedge s.isOptional = true) \\ \Rightarrow a.isOptional = true \quad (2)$$

At runtime, this constraint helps the designer in making choices: when an optional activity A depends on an optional artifact status S and when the designer choose not to produce this artifacts (i.e. he chooses not to realize the optional activities producing S), then the system can infer that the activity A cannot be realized. The system can thus avoid A from the choices it proposes to the designer.

The third constraint (3) is used at runtime to measure if an activity can be started. This is also true for the first activity of the process : it cannot input artifacts (nothing has been done at this time).

These constraints also define the order in which the activities can be realized, and thereby the order in which the goals can be chained.

$$\forall a1 \in Activity, \forall s \in Status, \\ (s \in a1.input \wedge a1.status = "running\ on") \\ \Rightarrow \begin{pmatrix} \exists a2 \in Activity, \\ s \in a2.output \wedge a2.status = "ended" \end{pmatrix} \quad (3)$$

The next section presents how M2Flex gives rise to creating flexible process models.

III. Flexibility In M2Flex

In this section, we detail how the process metamodel presented before helps defining process models that offer the various dimensions of the flexibility, as defined in [11].

We successively present how variability, completeness, granularability and distensibility are integrated into the metamodel.

A. Variability

Variability is defined as the possibility for the designer to make choices in a set of equivalent variants. This is supported at two levels of abstraction in M2Flex: on one hand, the different strategies leading from one goal to another offer different paths among which the designer selects one. On the other hand, at activities level, the Choice operator makes it possible for the designer to choose the activity he prefers. Implicitly, as mentioned before, this results in proposing various equivalent artifacts and/or roles.

M2Flex makes it possible for a process model implementing it to be variable, but it does not imply that this process model is variable: the multiplicity of the strategies associated to a goals pair is 1..* and there is no constraint that the process model offers any choice between strategies or activities.

It would be possible to add a constraint to make variability mandatory. But, as we wish to be able to represent various existing process models, we did not add this constraint.

B. Granularability

As mentioned in the section about activities, an elementary activity can be refined into other activities and elementary activities can be grouped into an *Activity*. This makes it possible to refine activities with different amounts of details or with different languages, e.g. an expert vocabulary at the level with the lowest amount of details and common language for novice designers when there are lots of details.

C. Completeness

Our metamodel implements the possibility of choosing a strategy among the others. However, this not enough to

implement the completeness as defined in [11]. Indeed, this definition has to be interpreted here as: (1) is it possible not to realize an activity in a selected path and (2) to pick up a default result in a repository instead of creating it.

The answer in M2Flex is given by the Choice operator. Indeed, as specified before, this operator makes it possible for activities to be optional, which is the first requirement of the completeness.

It also makes it possible to create a choice between an activity for creating the result - e.g. a users model - and an activity for picking a default result - e.g. a standardized users model - in a repository. Of course, in this case, the activity involving the default model will be realized, but not the creation of the result itself, which matches the second requirement of the completeness.

D. Distensibility

Activities also give rise to the last flexibility dimension, distensibility: a process model is said distensible (at enactment-time) when it includes well defined procedures for adding or avoiding components (activities, artifacts, roles,...). According to this definition, distensibility is not part of M2Flex itself: this property is verified when procedures accompany the metamodel. We have defined the required procedures.

The procedure for adding an activity A (and thereby the artifacts) is here:

- Instantiate the new activity A, associate it with (if needed new) statuses of artifacts and strategies
- Verify that all activities producing the statuses used as input in A exist (constraint 1)
- If one of these activities is missing, create it.

The procedure for avoiding an activity A (and thereby its associated artifacts) is:

- For each of the input statuses of A, consider if another activity uses it. If not, it is possible that the artifacts with these statuses are no more useful in the process and that the activities producing them could be avoided too or adapted if they produce other artifacts ; the unneeded artifacts can then also be avoided
- Delete A
- Delete all activities whose inputs are the outputs (or part of the outputs) of A (this can be automated)
- Adapt all activities that input (1) some of the outputs of A and (2) artifacts produced by activities other than A to eliminate the need of A outputs.

M2Flex integrates the metaclasses and associations required for instantiating flexible process models. When conforming to the metamodel, process models may support variability, completeness, granularability and distensibility. The next section presents an example of such a process model.

IV. AN EXAMPLE OF FLEXIBLE PROCESS MODEL

UsiXML [25] is a model-driven design and development method for creating plastic User Interfaces (UIs). Plastic UIs are interfaces that have abilities to dynamically adapt themselves to their context of use. The context of use is defined as the triplet <User, Platform, Environment> [26]. Plastic UIs can then adapt themselves to either the user's, the platform or the environment characteristics. For instance, such UIs are able to take into account the programming languages available on the system they are running on, the screen size of the targeted device, the brightness of the room and so on. UsiXML offers several metamodels, a process model and many tools for creating plastic UIs. The approach relies on successive model transformations, starting from the models of (user's) Tasks, Platforms, Domain (representing business concepts) and so on. Required adaptations to the context of use are calculated at the application runtime and adapted transformations are then performed. These transformations produce successively more and more concrete models, ending with the generation of the executable Final UI. A reverse process is defined, making it possible to start from a concrete model and to abstract some of the models with higher level of abstraction, like the Abstract UI or the Task model, but not the Platform or the Domain model. Figure 6 represents the UsiXML development process proposed on the W3C website [27]. This figure shows the four main stages for transforming successively the task model in Concrete UI thanks to graph transformations (a formalism chosen in UsiXML to represent model transformation and dialog). On the right of the picture, the generative programs and the rendering (that materializes how a particular UI coded

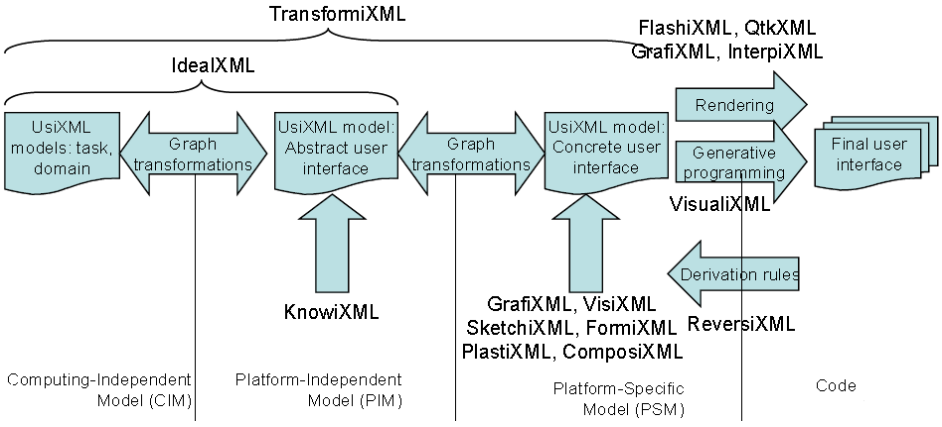

Fig.6 - UsiXML process model

in one language is rendered depending on the UI toolkit, the window manager and the presentation manager) produce an executable UI. Several tools can be used during the process, such as SketchiXML, which can prototype UIs.

UsiXML development process is poorly flexible [25]. Its variability is low, due to the small number of different paths in the process model: it is possible to start from the task model or to abstract it from a more concrete model. It offers no distensibility and partial completeness. For instance, it is possible to enter into the process at a more concrete level than task model (e.g. the concrete UI created using SketchiXML), but adaptability is then lowered and a reverse engineering of the task model is needed to ensure all plastic abilities. Finally, its granularability also is low, for it offers a very small number of levels of details and the descriptions of activities are written only in expert language.

We have studied this process model in order to make it (more) flexible using our meta-model. Several possibilities emerged during this work. Figure 7 shows three subsets of the UsiXML process model once made flexible. The graphical syntax shown on this figure will be detailed in section V, we focus here on the flexibility represented in the figure. The top of the figure (part a) presents the goals and the strategies. The second part of the figure (b) presents choices between activities and the last part of the figure (c) shows an activity refinement.

In the following, we present some of these new possibilities, grouped by flexibility dimension.

A. *Variability*

In order to make the process model able to propose various paths and especially paths that would offer a lower threshold of use [28] and/or increased possibilities of reusing already existing components, we have proposed several ways for generating the models instead of creating them.

Several strategies and choices between equivalent activities have been added in the process model.

- The task model can be generated by Compose [29], a framework that uses automated planning algorithms for generating a task model corresponding to a goal expressed by the user (or a designer in our case). This is achieved in the process model by adding a choice between the UsiXML "create task model" activity and a new activity "generate task model from Compose" (fig. 7b. activity 6[2]) in the strategy "model tasks, domain, environment" (fig. 7a. strategy 3).
- In the same strategy also, we defined a process for converting existing UIs into UsiXML models that can be abstracted into task model. This has been deeply studied on a concrete case in nuclear plants control command. In the process model, this corresponds to a new choice operator with a new optional activity, "Generate models from existing UIs".
- Similarly, UIs can now be drawn as mockups and converted into UsiXML concrete UI. We have implemented a complete solution starting from Balsamiq Mockups, a tool available on the Internet.
- We also created alternative activities in the "code Task2AUI", "code AUI2CUI" and code "CUI2FUI" strategies (fig. 7a). These new activities propose elementary transformations rules that can be picked up in a repository and then combined to create complete transformations. This contributes to make the process model easier to execute, because large parts of the needed transformations are now proposed by the system. The designer only has to select the interesting ones and to eventually adapt them.

B. *Granularability*

UsiXML did not offer the various levels of details nor the languages addressing various designers' expertise. These possibilities have been added in two different ways.

First, the creation of a process model conforming to our metamodel offers the possibility of defining activities refinements. We have added several refinements to activities, for instance a step by step explanation for configuring and executing our tool that generates the Domain model from a database (Fig. 7c). Secondly, defining alternative activities such as the integration of Balsamiq Mockups makes it possible for designers to produce a UsiXML model without having to learn it. Therefore, novice designers can now enact UsiXML process, while UsiXML experts can still realize the classical activities. This corresponds to the goal of the granularability: offering to designers activities with various level of complexity, corresponding to the designers' various expertise.

C. *Completeness*

As already mentioned, UsiXML offers no possibility of making some activities or artifacts optional. However, every designer does not want to produce fully plastic UIs. Sometimes designers want to address users' disabilities, sometimes they target several platforms and sometimes they want to take environment brightness or noise into account. They rarely want to address all of it at the same time, and even more rarely in the first version of their UIs. This is why we improved UsiXML abilities for managing incompleteness.

We have defined some default and optional models that designers can pick up in a repository when they do not need to address very specific issues in a model. We proposed a default user model, in which users are considered monolingual, able-bodied, and competent, some default platforms models, representing for instance PCs and Android Smartphones with Internet connections, and a default environment model, the environment being seen as “average”, i.e. neither too much or too less dark or luminous or noisy.

[2] The numbers in brackets refer to the numbers on figures 7 and 8.

To offer these possibilities, we added new activities in the process model ("pick up default users / domain / environment / platform model"), making it possible for the designer either to create each of these models or to use the proposed default models.

D. *Distensibility*

UsiXML offered a predefined and fixed set of activities. We modeled UsiXML process into one of our tool (see next section), representing all activities and needed artifacts. Thus, this process model takes benefit of the constraints, making distensibility possible.

After presenting how a quite rigid process model has been flexibilized thanks to the variability, granularability, completeness and distensibility modeled in M2Flex, we in the following section the tools supporting our metamodel.

V. TOOLS SUPPORTING THE METAMODEL

A lot of tools or functionalities are required to put M2Flex in action, at design time as well as enactment-time. We aim to create tools for designing a process model as an instance of our metamodel and to enact this process model while allowing designers to adapt it. This means that the process model has to be in turn instantiated as a process and that this process supports the flexibility, offering the possibility of selecting the activities, recording these choices and their motivations, showing their impacts of on the following possibilities. We also aim that the enactment of the process model impacts the development tools, for instance by configuring some development tools according to the choices of the designer.

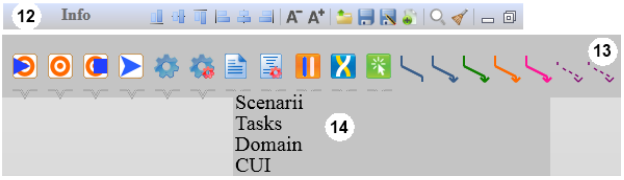

Fig.8 - D2Flex toolboxes

We have created D2Flex, a tool supporting the design of process models. Figure 7 has been created by assembling three screenshots of D2Flex. We can see examples of a Start goal (1), of casual goals (2) and of strategies (3). Activities (4) are concretizing (5) these strategies and they input (6) or output (7) artifacts, that can be documents (8) or executables. Refinement also is implemented: refined activities are represented with a "+" sign drawn of them (9). When displaying a refined activity, a new window is displayed, e.g. parts (b) and (c) on figure 7. It is then possible to design how the activity is detailed. On part (c) of figure 7, the activity "Generate [Domain model] from database" is detailed by two (sub-)activities, respectively "collect database access data" and "run DB2domain". This figure also shows an example of the "parallel" operator is presented in (10) and the "choice" operator in (11).

Figure 8 shows the two toolboxes offered by D2Flex. The first one (12) offers files managements and presentation functionalities (elements alignment or font size management, for instance). The second toolbox (13) shows the various

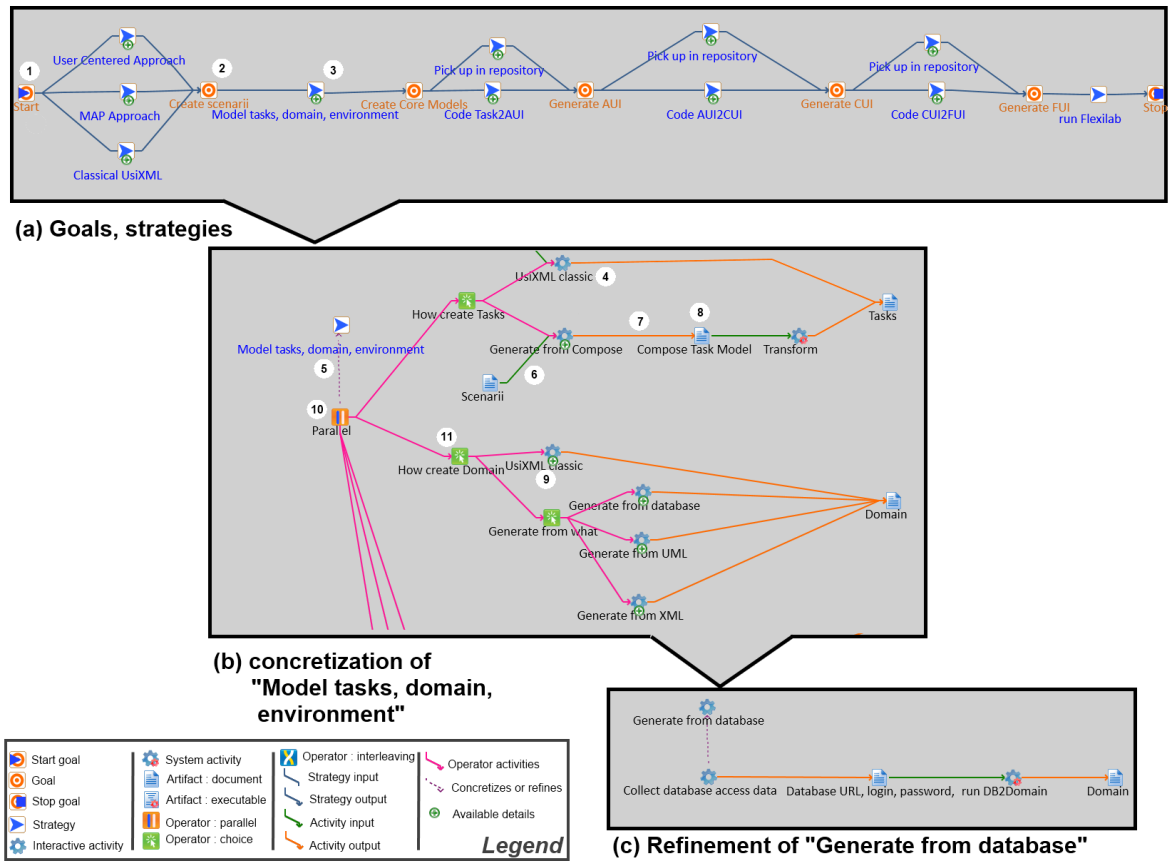

Fig.7 - UsiXML flexible process model

elements of the metamodel that can be instantiated in the process model. The list of already instantiated components (14) makes it possible to represent several times the same component in the process model. For instance, on figure 8, some already instantiated artifacts are proposed: they can be added in an activity refinement, letting the system know that both the representations refer to the same artifact.

We also have integrated some of the possibilities offered by flexibility in some tools related to UsiXML [25]. Therefore, we created a framework, whose first version is named UsiComp [30] and whose ongoing version is named FlexiLab. This framework offers two modules, one dedicated to the design of models and transformations, and the other for executing the transformations and producing the executable UI. This framework implements a part of the process model described in section IV (integration of Balsamiq Mockups, generation of the task model by Compose [29], repository of elementary transformations and default models and so on). However, there is (up to now) no link between D2Flex in which we describe the process and UsiComp/Flexilab in which we execute the resulting application. It is also not yet possible to have information about the process and adapt the process model implemented in UsiComp/FlexiLab at enactment-time.

In the next section, we compare our proposition with related works.

VI. Related Works

We have analyzed many approaches, process models and metamodels to define the four dimensions of flexibility (variability, granularability, completeness and distensibility). This study also relies on more than 500 scientific and industrial papers. This work led us to identify four main approaches in researches about flexible process models:

(a) "classical" approaches including or not some flexibility, such as the Spiral Model [31], Scrum [15] or FDD [32]. These approaches focus on a facet of the process model (activities, products, decisions, context, strategies) to elaborate the activities to be realized by designers and developers. The three examples mentioned before are activity-oriented, an approach whose metamodel is SPEM [33].

(b) approaches based on method engineering [34, 35, 36], where methods fragments, or chunks, are assembled to create an ad-hoc process model.

(c) studies on the integration of business process qualities and design process models [9]. These works intend to combine the expressiveness, understandability and abstraction of Software Process Modeling Languages with the capacity of executing processes of business processes.

(d) services-based approaches [37], which propose to build dynamically ad-hoc process models thanks to services representing methods fragments

In the following, we compare the flexibility provided by our process metamodel to the flexibility offered by four other approaches, that we consider being representative of the four categories mentioned before.

A. *"Classical" Approaches*

Software and Systems Process Engineering (SPEM) [33] is a standardized activity-oriented process metamodel proposed by the Object Management Group.

Activities are submitted to very generic conditions that enable a method engineer to specify the artifacts (and their states) required by activities. However, there is no structure making it possible to specify a free choice between equivalent activities or artifacts. There is no goals and strategies management: the offered variability is partial.

Conversely, its granularability is well established: an activity is modeled as a *BreakdownElement* and as a composition of others *BreakdownElements*. SPEM also offers a super-class named MethodPlugin that gives rise to more granularities. However, this class does not make SPEM distensible: methods plugins can only be added when designing the process model.

BreakdownElements can be optional. Similarly, a TaskDefinition may input optional *WorkProducts* (our artifacts). However there is no mechanism to define that a *WorkProduct* is pre-existing and can be picked up somewhere. SPEM completeness is therefore partial.

Another standard metamodel is given by ISO/IEC 24744 [38]. This metamodel uses powertypes (pairs of classes) such as *DocumentKind* (the kinds of documents described by the methodology) and *Document* (the documents that people manage). This makes possible to define that a task produces a *DocumentKind*, the choice of the concrete *Document* being made during enactment. Powertypes are used for defining artifacts as well as activities, giving rise to a high variability. The granularability is also high, thanks to several compositions/aggregation in process and artifacts definition.

Options are managed at several levels. For instance, the attribute named optionality in the *ActionKind* class represents the possibility for a task to use some *WorkProduct* or not. However, there is nothing like default results/artifacts that can be picked up on the shelf at enactment-time. The completeness of this metamodel is therefore partial. Moreover, ISO/IEC 24744 clearly states that its metamodel intended to be instantiated by method engineers (methodology domain), the resulting methodology being used by developers (endeavour domain). This is not what we aim to do in a distensible process (meta)model.

Other approaches exist. We already mentioned KAOS[17], I* [18] and the MAP [19] as examples of goal-oriented approaches. The Work Product Pool approach [39] is an example of a product-oriented approach: the authors focus directly on the products to be built. They reuse the notion of Work Product Kind defined in [38], making it possible to offer good variability in artifacts.

However, the authors leave the "*process*" (the activities) to be defined at enactment time. To achieve this, the Work Product Pool requires the "*assistance*" of a "*software tool*" for identifying the "*possible process/products*" pairs, i.e. the activities. The authors claim that, as the process "*is not part of the structure of the methodology, it can be freely changed as*

long as the overall product network stays constant". However, this makes poor native variability in activities, as they are not defined: the variability relies on external tools that are not guaranteed to exist and to offer variants. Therefore, the variability of the metamodel itself is partial.

A product can be considered at various levels of details and refined into inter-products. The Work Pool Product approach is thus granular. The authors explicitly say that an expected artifact can be either produced either found in the "reusable asset pool", thereby managing incompleteness.

The Work Product Pool approach defines a procedure for extending the process model at enactment time. It is thus distensible.

B. Situational Methods

A "*Process Engineering Method based on a Process Domain Model and Patterns*" is proposed in [34], aiming to help method engineers building process metamodels that suit the specific needs of their organization. This methods leads to build "*unified, fitted and multi-viewpoints process meta-models*". To achieve this, the authors have analyzed several process metamodels and propose an alignment of the various concepts coming from different metamodels, making it possible to assemble fragments of these metamodels. They enrich the resulting (meta-)metamodel with patterns that can be used to customize the selected process metamodel.

The domain model on which this approach relies offers two abstraction levels: (a) an intentional level representing the goals (named intentions) and (b) an operational level representing the activities that concretize these goals. It also offers the concept of *Alternative*, representing the various decisions that can be made when a problem is identified and that "*contributes to the advance of a Work Unit*" (a Work Unit being close to what we call here an activity). This concept offers possibilities of choosing an activity. These structures are equivalent to our proposition and offer the same possibilities regarding variability.

Activity refinement is not native in the metamodel: the *Work Units* (activities) have no sub-structure. However, the method defines a pattern for adding a reflexive composition or aggregation to a class. Therefore, it is possible to specify a composed structure as a customization. Granularability is thus possible.

This approach is made for building process metamodels and embedding all what a method engineer may need. It integrates thus all the mechanisms for adding elements to the metamodel. However, this is not the point we address when we analyze distensibility. Indeed, the metamodel is said distensible when it produces distensible process models. In this approach, there is nothing that makes it possible to extend or retract components at process enactment-time. This metamodel is therefore not distensible.

The metamodel proposes an attribute for managing options in the *WorkUnit* class, but proposes no default artifact. Its completeness is therefore partial.

C. Hybrid Business & Development Process Models

The UML4SPM to WS-BPEL approach [9, 40] combines UML4SPM, a metamodel for software process modeling and WS-BPEL, an XML-based standard supporting Web services orchestration in the context of business processes implementation. The former is expected to provide expressiveness, understandability and abstraction and the latter to offer the concepts supporting process execution.

Being based on UML 2.0 and SPEM, UML4SPM origin ensure him a good granularability. UML4SPM includes components (mainly the *Decision* and *Merge Nodes* classes) for expressing decisions and alternatives. Combined to the hierarchical structure, this can be considered as an equivalent to strategies. This metamodel is therefore variable.

Activities being constrained by generic pre- and post-conditions and by analyst's decisions, they may be optional. However, there is no mention of any available pre-defined results, such as a default (and thus simplified) users model. The completeness of this approach is therefore partial.

The approach relies on (a) the modeling of the process with UML4SPM, (b) the mapping of process model components into BEPL elements and (c) the execution of these elements as services. The approach would then be distensible if (a) the process model is distensible at runtime and (b) the services corresponding to new components were dynamically integrated into the orchestration. Even if it is possible to take into account the emergence of new services dynamically, the approach does not define the complete procedures for extending a process model. The distensibility is thus partial.

D. Service-based approaches

The Service-Oriented Meta-Method (SO2M) [41] proposes a model for composing dynamically methodological services that provide solutions for development problems. The services are goal-oriented so that to manage the knowledge for describing and solving a problem.

SO2M provides pre-defined services, among which designers and developers choose those that suit their needs. SO2M offers guidance for selecting the services. Moreover, the services are focusing on the problem more than the solution and the composition may dynamically provide various paths to solve the problem. The process metamodel is therefore variable.

A service may invoke others services when needed, offering thereby some granularity. However, there is no mention that equivalent services with different levels of details can be defined and composed. The granularability of the resulting process model is thus partial.

Designers and developers are invited to explore a set of pre-defined services and to select those that suit their needs. As services can be chosen or not at enactment-time, there is some management of completeness. However the completeness is partial, because there is no pre-defined results that the designers and developers could reuse. Moreover, even if the set of services is obviously extendable and even if services can

dynamically been added to an orchestration, there is no defined procedure for enabling the designers and developers to create new services at runtime. The distensibility is partial.

In addition to these approaches dedicated to software design and development, we have analyzed the flexibility offered by Business Process Model and Notation (BPMN). According to [42], BPMN supports high flexibility: variants can be modeled by proposing various paths. However, this does not make it possible to express anything like goals and strategies. Its variability is therefore partial. It manages granularability as well as distensibility, but, even if default paths (named Default Sequence Flow) are possible, BPMN does not mention default result that could be picked up on the shelf. The completeness also is therefore partial.

E. Synthesis

In summary, Table I shows that flexibility is only partially supported in the literature, especially in terms of completeness and distensibility at runtime.

TABLE I. FLEXIBILITY IN 8 APPROACHES

Approaches	**Kinds of flexibility**			
	Var.	***Gran.***	***Comp.***	***Dist.***
M2Flex	high	high	high	high
SPEM	partial	high	partial	none
ISO/IEC 24744	high	high	partial	none
Work Product Pool	partial	high	high	high
Process Engineering Method	high	high	partial	none
UML4SPM to WS-BEPL	high	high	partial	partial
SO2M	high	partial	partial	Partial
BPMN	partial	high	partial	high

VII. CONCLUSION AND PERSPECTIVES

This paper promotes flexibility of design and development process models, even at enactment time. The corner stone is M2Flex, a process metamodel that covers the four dimensions of flexibility: (1) variability, the ability of the metamodel to provide several equivalent choices, (2) granularability, the possibility of defining components with multiple levels of details, (3) completeness, the possibility of defining optional components and pre-defined reusable results, and (4) distensibility, the capacity of the resulting process model to be extended or cut at runtime.

M2Flex is original by the flexibility it offers to designers and developers, not only at design time as it is classically done, but also at runtime which is new to our best knowledge.

In addition to the metamodel, we present rules for validating the process model being built. We also describe an instantiation of M2Flex that shows that the resulting process model is understandable and usable. Last but not least, we present the tools we created for managing and implementing M2Flex.

In the future, we plan to improve our tools (for instance, with validity checkers for the constraints to be satisfied). We also plan to create additional tools like for instance a module for executing the process models compliant with M2Flex. Attention will be paid to distensibility and to impact development tools by configuring and/or executing them. We also intend to make extensions sharable and reusable: for instance, if an activity is created by a team, it might be made available to others. Finally, as soon as this series of tools will be available, we plan to evaluate usage and to collect best practices.

ACKNOWLEDGMENT

The authors warmly thank the European ITEA UsiXML project, which strongly supported this work.

REFERENCES

[1] F. Garzotto and V. Perrone, “Industrial Acceptability of Web Design Methods: an Empirical Study”, *Journal of Web Engineering*, vol. 6, no. 1, pp. 73–96, 2007.

[2] C. Barry and M. Lang, “A Survey of Multimedia and Web Development Techniques and Methodology Usage”, *IEEE MultiMedia*, vol. 8, no. 2, pp. 52–60, Apr. 2001.

[3] B. Fitzgerald, “An empirical investigation into the adoption of systems development methodologies,” *Information & Management*, vol. 34, no. 6, pp. 317 – 328, 1998.

[4] G. Booch, *Object-Oriented Analysis and Design with Applications*, 2nd ed. Addison-Wesley, 1993.

[5] Harmsen, “Situational Method Engineering”, University of Twente, Moret Ernst & Young Management Consultants, Netherlands, 1997.

[6] V. R. Basili and H. D. Rombach, “Tailoring the software process to project goals and environments”, in *Proceedings of the 9th int. conference on Software Engineering*, Los Alamitos, USA, 1987, pp. 345–357.

[7] C. Potts, “A generic model for representing design methods” in *Proceedings of the 11th int. conference on Software engineering*, New York, 1989, pp. 217–226.

[8] F. Harmsen, S. Brinkkemper, and J. L. H. Oei, “Situational method engineering for informational system project approaches” in *Methods and Associated Tools for the Information Systems Life Cycle*, 1994, pp. 169–194.

[9] R. Bendraou, A. Sadovykh, M.-P. Gervais, and X. Blanc, “Software Process Modeling and Execution: The UML4SPM to WS-BPEL Approach”, in *EUROMICRO-SEAA*, 2007, pp. 314–321.

[10] C. Hug, A. Front, and D. Rieu, “A Process Engineering Method based on a Process Domain Model and Patterns”, in *MoDISE-EUS*, Montpellier, France, 2008, vol. 341.

[11] E. Céret, S. Dupuy-Chessa, G. Calvary, A. Front, and D. Rieu, “A taxonomy of design methods process models,” *Information and Software Technology, Elsevier*, vol. 55, no. 5, pp. 795–821, May 2013.

[12] J. L. Kolodner, R. L. S. Jr, and K. Sycara-Cyranski, “A Process Model of Cased-Based Reasoning in Problem Solving,” in *IJCAI*, 1985, pp. 284–290.

[13] R. C. Schank, *Dynamic memory - a theory of reminding and learning in computers and people*. Cambridge University Press, 1983.
[14] M. Laanti, "Implementing Program Model with Agile Principles in a Large Software Development Organization," in *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*, 2008, pp. 1383–1391.
[15] K. Schwaber, "SCRUM Development Process," in *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA)*, 1995, pp. 117–134.
[16] J. Martin, *Rapid application development*. Indianapolis, IN, USA: Macmillan Publishing Co., Inc., 1991.
[17] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Science of Computer Programming*, vol. 20, no. 1–2, pp. 3 – 50, 1993.
[18] E. Yu, "Modelling Strategic Relationships for Process Reengineering," Department of Computer Science University of Toronto, Toronto, 1995.
[19] C. Rolland, N. Prakash, and A. Benjamen, "A Multi-Model View of Process Modelling," *Requirements Engineering*, vol. 4, no. 4, pp. 169–187, Dec. 1999.
[20] C. Atkinson and T. Kühne, "The Essence of Multilevel Metamodeling," in *≪UML≫ 2001 — The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, vol. 2185, M. Gogolla and C. Kobryn, Eds. Springer Berlin Heidelberg, 2001, pp. 19–33.
[21] J. McDermid and K. Ripken, *Life Cycle Support in the ADA Environment*. New York, NY, USA: Cambridge University Press, 1984.
[22] D. A. Norman and S. W. Draper, *User centered system design: new perspectives on human-computer interaction*. Lawrence Erlbaum Associates, 1986.
[23] L. Nóbrega, N. Nunes, and H. Coelho, "Mapping ConcurTaskTrees into UML 2.0," in *Interactive Systems. Design, Specification, and Verification*, vol. 3941, S. Gilroy and M. Harrison, Eds. Springer Berlin Heidelberg, 2006, pp. 237–248.
[24] Tech-IS, "Oxygen Code Generator." [Online]. Available: http://www.oxygencode.com/. [Accessed: 27-Oct-2012].
[25] Q. Limbourg and J. Vanderdonckt, "UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence" in *ICWE Workshops*, 2004, pp. 325–338.
[26] G. Calvary, J. Coutaz, and D. Thevenin, "A Unifying Reference Framework for the Development of Plastic User Interfaces," in *Engineering for Human-Computer Interaction*, vol. 2254, M. Little and L. Nigay, Eds. Springer Berlin / Heidelberg, 2001, pp. 173–192.
[27] W3C, "UsiXML - Model-based User Interfaces Incubator Group Wiki." [Online]. Available: http://www.w3.org/2005/Incubator/model-based-ui/wiki/UsiXML. [Accessed: 30-Jan-2013].
[28] M. Resnick, B. Myers, K. Nakakoji, B. Shneiderman, R. Pausch, T. Selker, and M. Eisenberg, "Design Principles for Tools to Support Creative Thinking" *A Workshop Sponsored by the National Science Foundation*, pp. 25–35, 2005.
[29] Y. Gabillon, M. Petit, G. Calvary, and H. Fiorino, "Automated planning for user interface composition" in *Proceedings of the 2nd International Workshop on Semantic Models for Adaptive Interactive Systems: SEMAIS'11 at IUI 2011 conference*, 2011.
[30] A. G. Frey, E. Ceret, S. Dupuy-Chessa, G. Calvary, and Y. Gabillon, "UsiComp: an extensible model-driven composer" in *EICS*, 2012, pp. 263–268.
[31] B. Boehm, "A spiral model of software development and enhancement" *SIGSOFT Softw. Eng. Notes*, vol. 11, pp. 14–24, Aug. 1986.
[32] P. Coad, E. Lefebvre, and E. DeLuca, *Java Modeling in Color with UML: Enterprise Components and Process*. Upper Saddle River, NJ: Prentice Hall PTR, 1999.
[33] Object Management Group, "Software & Systems Process Engineering Metamodel (SPEM)." Apr-2008.
[34] C. Hug, A. Front, and D. Rieu, "A Process Engineering Method based on a Process Domain Model and Patterns," in *International workshop MoDISE-EUS 2008 (Model Driven Information Systems Engineering: Enterprise, User and System Models), held in conjunction with CAISE 2008, Montpellier, France*, 2008.
[35] I. Mirbel and V. De Rivière, "Introducing flexibility in the heart of analysis and design," presented at the 6th World Multi-conference on Systemics, Cybernetics and Informatics (SCI 2002), Orlando, USA, 2002.
[36] B. Henderson-Sellers, C. Gonzalez-Perez, and J. Ralyté, "Comparison of Method Chunks and Method Fragments for Situational Method Engineering," in *Australian Software Engineering Conference*, 2008, pp. 479–488.
[37] N. Arni-Bloch and J. Ralyté, "Service-Oriented Information Systems Engineering: A Situation-Driven Approach for Service Integration" in *CAiSE*, 2008, pp. 140–143.
[38] ISO/IEC, "Software Engineering — Metamodel for Development Methodologies." International Organization for Standardization, 2007.
[39] C. Gonzalez-Perez and B. Henderson-Sellers, "A work product pool approach to methodology specification and enactment," *Journal of Systems and Software*, vol. 81, no. 8, pp. 1288–1305, 2008.
[40] R. Bendraou, M.-P. Gervais, and X. Blanc, "UML4SPM: A UML2.0-Based Metamodel for Software Process Modelling," in *Model Driven Engineering Languages and Systems*, vol. 3713, L. Briand and C. Williams, Eds. Springer Berlin / Heidelberg, 2005, pp. 17–38.
[41] G. Guzelian, "Conception de systèmes d'information, une approche orientée service," PhD dissertation, University Paul Cézanne, Marseille, France, 2007.
[42] M. Cortes Cornax, A. Matei, E. Letier, S. Dupuy-Chessa, and D. Rieu, "Intentional Fragments: Bridging the Gap between Organizational and Intentional Levels in Business Processes" in *OTM Conferences*, 2012, pp. 110–127.