# Providing Flexible Process Support to Project-Centered Learning

Stefano Ceri, Florian Daniel, Maristella Matera, and Alessandro Raffio

**Abstract**—While business process definition is becoming more and more popular as an instrument for describing human activities, there is a growing need for software tools supporting business process abstractions to help users organize and monitor their desktop work. Tools are most effective when they embed some knowledge about the process, e.g., in terms of the typical activities required by the process, so that users can execute the activities without having to define them. Tools must be lightweight and flexible, so as to enable users to create or change the process as soon as there is a new need. In this article, we first describe an application-independent approach to flexible process support by discussing the abstractions required for modeling, creating, enacting, and modifying flexible processes. Then, we show our approach at work in the context of project-centered learning. In this application, learners are challenged to perform concrete tasks in order to master specific subjects; in doing so, they have to conduct significant projects and cope with realistic (or even real-life) working conditions and scenarios. Often, students are geographically dispersed or under severe timing constraints, because these activities intertwine with their normal university activity. As a result, they need communication technology in order to interact and workflow technology in order to organize their work. The developed platform provides a comprehensible, e-learning-specific set of activities and process templates, which can be combined through a simple Web interface into project-centered collaboration processes. We discuss how the general paradigm of flexible processes was adapted to the learning concept, implemented, and experienced by students.

**Index Terms**—Web-based teamwork processes, process design, flexible processes, e-learning, Web application design.

## 1 Introduction

Workflow Management Systems (WfMSs) support the definition and the execution of business processes. They are based on process models that, at design time, capture constraints on the execution of tasks as well as their assignment to users and their mapping to resources. Then, WfMSs manage task enactment at runtime according to the model. Workflow applications are typically well defined and highly repetitive in nature, so that the process definition is a specialized task by itself, performed by a process modeling expert by using a sophisticated tool environment; once the process is defined, the system normally supports several concurrent process instantiations (or cases), each one characterized by its own set of input parameters, users, and resources.

While the explicit definition of business processes is becoming more and more popular in the early phases of software engineering, WfMSs cannot claim a comparable success and popularity: many applications, which do rely on well-defined process models, are however still implemented by means of conventional software. This lack of success, on the one hand, is motivated by the complexity of the WfMSs themselves and by the spread of the market of workflow management products, which is characterized by a large number of different products without a clear market leader [32]; on the other hand, the excess of rigidity of workflow enactment leaves no freedom to users. There is however a growing need for more flexible tools, capable of giving design control to users who are interested in modifying, adapting, and extending existing process definitions autonomously. In line with these considerations, this paper is focused on *flexible process management support*, i.e., the specification and execution of processes in contexts where the full process specification is determined by the actual actors of the process when they approach process execution or even while the process is already executing.

We have been motivated to design and deploy a new paradigm for flexible processes by an earlier experience in supporting project-centered learning through conventional processes. More precisely, our research has aimed at assisting teams of master students who are challenged to perform concrete learning tasks about given subjects in a structured and collaborative fashion. In such collaborative environments, coordination processes are typically the result of consensus decisions obtained after discussions among the team members. Processes that are performed by different teams may therefore be different. Even if processes are based on a common core (or *template*), they usually need to be adapted to the specific team's collaboration style. For instance, different teams may require processes that differ in their timings and deadlines, or they may need to add activities or verification cycles to the original template, depending on the skills and maturity of the team components and of the team as a whole.

Initially, we expected that teams would have made use of "static" processes in form of Web applications [7], but we

- *S. Ceri, M. Matera, and A. Raffio are with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Via Ponzio 34/5, 20133 Milano, Italy. E-mail: {ceri, matera, raffio}@elet.polimi.it.*
- *F. Daniel is with the Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento, Via Sommarive 14, 38100 Povo (TN), Italy. E-mail: daniel@disi.unitn.it.*

*Manuscript received 1 July 2007; revised 9 Jan. 2008; accepted 16 June 2008; published online 25 June 2008.*
*Recommended for acceptance by Q. Li, D. McLeod, R. Lau, and J. Liu.*
*For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2007-07-0313.*
*Digital Object Identifier no. 10.1109/TKDE.2008.134.*

soon realized that such approach had not enough flexibility. Static processes partially help in the early or final phases of the work (e.g., team formation, delivery of the team's final report for review), but even in these cases team-specific changes would be needed. We also understood that graduate students coworking in projects could not act as designers of their static processes, regardless of any technological support that we could design for them. Students have a limited IT background and a very limited availability in experiencing new technologies, as they are anyway involved in a very heavy workload for completing their masters.

After realizing that our approach could not be used as is, we stepped back and tried to understand from scratch the requirements for a new and effective concept of flexible project. This work has finally led to the development of a collaborative, open environment for project-centered learning—the *COOPER* platform—which heavily leverages on the idea of flexible, user-centric process support. The approach, described in this paper, is characterized by the following main contributions:

- We propose an intuitive process definition logic that relies on a predefined set of *customizable activities*; in cooperation with pedagogical experts, we have been able to identify about 40 different activity types (see Section 3.2), covering the specific needs of the cooperative teamwork context. Activities are described by means of parameters (e.g., the max number of students allowed to take place in a synchronous videoconference), with powerful defaults so as to keep at a minimum the parameters that need to be specified by users. Activities have a predefined implementation that includes their Web interfaces and supports the recording of the activity's state and of possible side effects (e.g., an activity is completed, a document is downloaded).
- We provide an easy-to-use user interface enabling even inexperienced users to specify workflows in a *controlled fashion* by means of a form-based composition mechanism oriented toward users. In order to keep the process definition simple, we have sacrificed some expressive power.
- We allow the user to *explicitly control* the progression of a workflow by means of control points that are represented by decision activities to be performed by the user. The interaction paradigm thus always presents users with all possible legal activities that can be performed after the completion of a given activity and, based on the user's choice, the workflow progresses accordingly. The motivation of this design principle is to put the user in charge of driving the workflow at all stages, leaving no hidden semantics. In practice, this principle allows us to limit the complexity of conditions: the proposed dynamic processes have indeed only very few decision variables, typically required to control activity enactment in presence of conjunction, disjunction, and loops, whose value is sufficient for taking simple decisions about the next activities to be proposed to users.
- We enable the *independent evolution and modification* of process definitions, even during process execution. Processes may derive from an initial template, but teams are enabled to change them regardless of other process instances descending from the same template. By resorting to a terminology that came into use with OEM [25] and become then popular with XML, we say that our processes are *self-describing*, meaning that they carry with them not only their state representation but also their schema description. The evolution of self-describing processes is much simpler than that of template-depending processes (see, for instance, [11]).
- We enable *accountability*, i.e., the full a-posteriori tracing of process execution, in terms of a state describing the performed activities, with indications about timings, users, resources being used, and their associated state.

In summary, we believe that flexible processes can be enabled through customized activities, a careful use of form-based design, and explicit workflow control; this, as we will formally discuss in this paper, partially reduces the class of processes that can be generated. However, portability and accountability are guaranteed much in the same way as with static processes, and independence is added as a feature that greatly simplifies process evolution. In the next section, we discuss how the above design principles were transformed into modeling concepts and system specifications, which are independent of the specific customization domain. We will then describe the application of the resulting framework to the domain of collaborative e-learning.

## 2 A Model for Flexible Processes

The concept of flexible processes addressed in this research leverages on two main ideas: the need to provide process designers with a strong guarantee of the *semantically correct execution and termination* of process instances, and the possibility to easily (flexibly) *modify* processes even during runtime. Providing guarantees on the process semantics aims at assisting the continuous redefinition or evolution of running processes by users that in most cases are inexperienced.

### 2.1 Process Model

As opposed to traditional workflow products, our approach provides domain-specific support for the organization of work into process structures. We therefore start with defining the concept of *activity type*, which is one of the cornerstones of the proposed approach to flexible processes.[1]

**Definition 1 (Activity type).** *An* activity type *is defined as a tuple* $T = \langle Name, HT, P \rangle$, *where*

- $Name$ *is the name of the domain-specific activity type.*
- $HT$ *is the activity type's hypertext front end, which is used as user interface for the execution of the activity*

1. In general, the term *activity* denotes a concept of the workflow model, while the term *task* denotes the execution of a portion of work by the user.

*(each time an instance of a particular activity type is executed, the user is provided with the predefined hypertext portion).*

- $P = \{p_i\}$ *is a set of activity-specific properties, such as description, deadline, etc., to be configured during the process definition when the activity type is instantiated.*[2]

With the help of typed activities, modeling even complex working scenarios is highly facilitated, provided that each task that can be performed has already been defined through a proper activity type. A process designer, hence, needs to have access to a *library of activity types*, which can be domain- or application-specific. The library needs to be as complete as possible in its particular domain, i.e., all possible activities that may be required during process execution need to be recognized, in order for the library to represent a valuable instrument in the hands of the process designer. Given that the library of available activity types depends on the domain, it is not possible to predefine an exhaustive set of universal activity types in advance; as a matter of fact, activity types are known at system configuration time and depend on the interaction between the IT expert who is configuring the system and the domain expert who is aware of the users' needs in terms of the activity library.

The definition of a process requires thus the instantiation of a set of activities. We distinguish between two kinds of activities: *user activities* and *system activities.* The former are instantiations of activity types as defined above, the latter represent routing activities, which can be used to specify the structure of the process, rather than the single work items. User activities are defined as follows:

**Definition 2 (User activity).** *A* user activity *is the instantiation of an activity type $T$, which corresponds to a task to be performed by a user during process execution, and is defined as: $A^U = \langle Type, Name, PV, State \rangle$, where*

- $Type$ *is the name of the instantiated activity type,*
- $Name$ *is the name assigned to the activity during process definition,*
- $PV = \{\langle p_i, vi \rangle\}$ *are the values $v_i$ for each property $p_i$, and*
- $State$ *is the state of the activity during process execution (see Section 2.3 for further details).*

System activities are defined as follows:

**Definition 3 (System activity).** *A* system activity *is executed by the system and can be defined as $A^S = \langle Type, Name, PV, State \rangle$, where*

- $Type \in \{Start, End, AndSplit, OrSplit, AndJoin, OrJoin, LoopSplit, LoopJoin\}$,
- $Name$ *is the name assigned to the activity,*
- $PV = \emptyset$ *(system activities do not have properties), and*
- $State$ *is the state of the activity during process execution (see Section 2.3 for further details).*

The *Start* and the *End* activities denote the beginning and the end point of a process definition, respectively. All the other types denote routing points, which are needed to structure the process flow. The *AndSplit* allows the creation of parallel execution flows, synchronized by means of an *AndJoin.* The *OrSplit* allows the creation of alternative process flows, where only one flow can be activated for execution. The *LoopSplit* and the *LoopJoin* allow the definition of cycles. The semantics of system activities is standard, see, e.g., [32]; our main choice was adopting an exclusive semantics for the *OrSplit*.

The instantiation of a user activity requires the association of the activity with the users that will execute it. Users may pertain to user groups.

**Definition 4 (User group).** User groups *represent clusters of users that express the roles that users may play in the system. Each user may belong to one or more user groups.*

The instantiation of a user activity may also involve the association of the activity with resources.

**Definition 5 (Resource).** *A* resource *is an instance of data (e.g., a tuple in a database or a file on the hard disk) that can be associated with one or more activities for inspection or manipulation.*

Given the previous definitions, the process model our approach relies on can be summarized as follows:

**Definition 6 (Process model).** *A* process model *is defined as the tuple $PM = \langle Name, A, R, U, succ, res, act \rangle$, where*

- $Name$ *is the name of the process;*
- $A$ *is the set of user and system activities in the process;*
- $R$ *is a set of document resources associated to user activities ($R$ may be empty);*
- $U$ *is the set of users participating in the process execution, possibly grouped by role—each user activity is associated either with one individual user (meaning that only the specified user is required to execute the activity) or with a user group (meaning that any user of the group may execute the activity);*
- $succ$ *is the function that allows the construction of the (directed) process graph by associating each activity in $A$ with one, more, or no successor activities (user activities always have one successor, system activities may have one or more successors, with the only exception of the End node that has no successor*[3]*);*
- $act$ *is the function that associates each user activity either with an individual user or with a user group; and*
- $res$ *associates resources with user activities.*

## 2.2 Process Creation

Definition 6 leads to the specification of arbitrary process models, based on the notion of activity types. The aim of our framework is, however, to guide users in the definition of correct process models. This can be partially achieved through the availability of the predefined activity type library. In addition, we propose the adoption of a simple set of fixed modeling constructs that can be recursively composed in order to achieve arbitrarily complex process logics:

2. Each activity type has a very own set of properties specified at activity type definition, which cannot be anticipated in this general definition.

3. The semantics of the function $succ$ depends on the standard semantics of system activities, see [32].

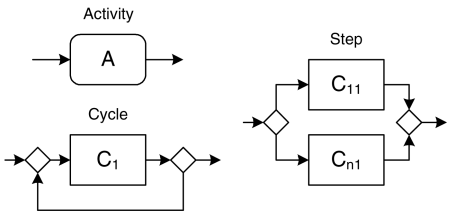



Fig. 1. High-level process modeling constructs in the guided process definition procedure. Rounded boxes are atomic activities, squared boxes are constructs that recursively may contain other constructs.

**Definition 7 (Modeling construct).** Modeling constructs *are particular modeling configurations that are composed of user and system activities and, recursively, of other modeling constructs. As illustrated in Fig. 1, allowed constructs are*

- *single* user activities,
- steps *of parallel branches, which are opened by a system activity of type* OrSplit *or* AndSplit *and closed by the corresponding join system activity (the* OrSplit *implies the execution of only one of the parallel branches of a step, the* AndSplit *implies the parallel execution of all the branches), and*
- cycles, *which are opened by a* LoopJoin *system activity and closed by a* LoopSplit *system activity.*

The previous modeling constructs enable building complex process logics, by adopting the following construction rules:

- Each process must start with a Start activity.
- Process definition proceeds by iteratively adding new constructs after the last inserted construct, until the process definition is completed. More precisely, the insertion of new constructs is allowed only after a Start node or after any of the allowed constructs.
- The addition of a new construct may lead to the following situations that depend on the construct type:
  - If the new construct is an *activity*, the new activity is concatenated to the last construct in the process definition.
  - If the new construct is an *And/Or-step*, the opening split is connected to the last construct in the process definition, and a corresponding closing join is added. The step definition then proceeds by constructing each branch by the arbitrary concatenation of the allowed constructs. After the definition of the branches of the step, the step definition is closed.
  - If the new construct is a *cycle*, an opening LoopJoin and a corresponding LoopSplit are inserted, and the body of the cycle is constructed by arbitrarily concatenating allowed constructs. Once the body of the cycle is defined, the cycle is closed.
- Each process must end with one *End* node.

**Lemma 1.** *The process model resulting from the application of the previous modeling rules is a* structured *process model, as defined in [19].*

**Proof.** We note that the composite modeling constructs allowed in our process model (cf. Fig. 1) comply with the basic workflow models in [19]: the *Or-step* is equivalent to the *Decision structure*, the *And-step* is equivalent to the *Parallel structure*, and the *Cycle* is a simplified version of the *structured loop*, in which we only use the repeat-until configuration.

Analogously as in [19] and based on the previous construction rules, we can thus inductively say that

- a process consisting of a single *activity* is structured,
- the *concatenation* of two structured processes is structured,
- the *Or-step* and the *And-step* of structured processes are structured, and
- the *Cycle* having a structured process as body is structured. □

**Theorem 1.** *A process model constructed according to the previously described construction rules is* well formed, *i.e., it never leads to a deadlock, nor to multiple active instances of the same activity.*

**Proof.** We observe that a generic process model will *deadlock* if the branches of an OrSplit are joined by an AndJoin, and that a generic process model may lead to *multiple instances* of an activity that are active at the same time if the branches of an AndSplit are joined by an OrJoin. In Lemma 1, we showed that process models constructed according to the described construction rules are *structured.* Hence, splits and joins are coupled correctly by construction, thus effectively avoiding situations of deadlock or multiple running instances of a same activity. □

It is worth noting that our process model is *lightweight* in that it does not require any process variables to guide the control flow at runtime; conditions over process variables are thus not necessary, and this eases the comprehension of a process model and yields a process semantics that is more accessible to users who are not familiar with process modeling. The state of a running process is therefore determined by the progression of the workflow and the contents produced by its users.

This design decision depends on the peculiar nature of collaborative processes addressed by our framework, which are as a matter of fact *user-driven.* Namely, users are the driving force that controls the process flow. For instance, if there is an OrSplit routing node in the process definition, we do not base the routing decision on the evaluation of a predicate over process variables, but instead we consider explicit user decisions: the user decides which branch to execute by simply starting one of the ready activities. Once an activity of one branch has been chosen, activities of the other branch are disabled, i.e., they are no longer executable. The same holds for cycles: after each execution of the cycle's body, an explicit user input decides whether to iterate or to leave the cycle.

### 2.3 Process Enactment

Up to now we discussed the *static* properties of our process model. In the following, we define its *dynamic* properties,

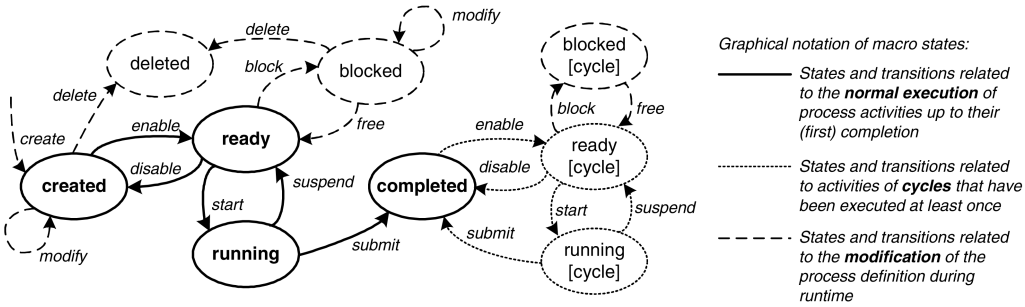



Fig. 2. Life cycle of instantiated (created) activities in a dynamically changing process definition. States in the state chart diagram represent the phases that an activity undergoes in its life cycle, transitions represent system or user actions. For the description of the states and the transitions, see Tables 1 and 2.

which refer to the execution phase and turn the so far static process model into a dynamic process model.

Dynamic properties represent a distinguishing feature of the proposed model with respect to other proposals discussed in the literature [11]. In fact, our approach overlaps the ideas of process model and process instance.[4] Each process definition has only one executing instance or, the other way around, each process execution has its own process definition. This interpretation of the process leads to *self-describing processes*: A process definition bears all *metadata* (process structure, role assignments, resource assignments) and *runtime data* (state of the process, states of activities, execution time stamps) necessary for the correct execution and for the monitoring of the process. Self-describing processes allow us to support process modification operations (see Section 2.4).

Fig. 2 describes the legal execution states of an activity in a running process. The state chart diagram is composed of three kinds of states, each corresponding to a different execution mode of the activity: normal execution, cyclic, and modification mode. The *normal execution* mode (in boldface) includes all standard states of activities in a running process; the *cycle* mode (in dotted lines) includes the states of activities that are part of a cycle and have already been executed at least once;[5] and the *modification* mode (in dashed lines) includes the states into which activities are automatically entered when process execution is halted because a new process definition activity takes place. Table 1 summarizes the meaning of the states an activity may assume during its life cycle.

Table 2 summarizes the transitions that determine state changes in Fig. 2. Transitions correspond to operations which are performed either by the user or by the system. Accordingly, we distinguish between *user operations* (i.e., *start*, *suspend*, *submit*, *delete*, *create*, and *modify*) and *system operations* (*enable*, *disable*, *block*, and *free*). The user operations *start*, *suspend*, and *submit* reflect the typical actions a user may perform in a WfMS during runtime, while the operations *delete*, *create*, and *modify* show the actions the user may perform during the runtime process modification (see the next section). Analogously, the system operations *enable* and *disable* refer to the execution of a process, while the operations *block* and *free* refer to the transition from execution mode to modification mode and vice versa.

The logic of Fig. 2 can thus be summarized as follows: during the execution of a process, activities are either in the normal execution mode or in the cycle mode. If a process designer starts the modification mode, already completed activities cannot be altered anymore; only activities in the *created* and the *ready* state may be modified. In order to prevent inconsistencies between the process under modification and its execution, no activities are enabled during the modification, and activities in the *ready* state get *blocked*; running activities may be completed without impacting the process definition. When modifying a running process, a process definer may thus delete or modify *created* or *blocked* activities only, or she/he may create new activities.

After the process modification, *blocked* activities are *freed* (i.e., they are again *ready* for execution), and the normal execution of the process proceeds according to the possibly modified process description.

## 2.4 Process Modification

As described in Fig. 2, during process modification, a process definer may *create*, *delete*, or *modify* activities. The existence of such modification operations at runtime is the essence of flexible processes. To prevent possible inconsistencies during process modification, we propose a set of process editing operations that respect the idea of "correctness by construction," based on the modeling constructs already

TABLE 1
Description of the Activity States

| Exec mode | State | Description |
|---|---|---|
| Normal execution | created | The activity is instantiated as part of the process definition. |
| | ready | The activity is ready to be run (from the user's to-do list). |
| | running | The user is working on the activity. |
| | completed | The activity has been completed. |
| Cycle | ready [cycle] | The activity is ready to be run. As part of a cycle, it has already been executed once. |
| | running [cycle] | The activity is running. As part of a cycle, it has already been executed once. |
| Modification | blocked | The modification mode is active. Ready activities may be modified or deleted. |
| | blocked [cycle] | The blocked and already executed activity of a cycle cannot be modified or deleted. |
| | deleted | The activity has been deleted during a modification of the process model. |

4. As we will show in Section 2.5, we also allow the definition of process templates, which play the role of process models.

5. We need to distinguish activities that are completed, but will not be executed anymore (*normal execution* mode), from activities that are completed, but could be executed again (*cycle* mode), in order to guarantee correctness during process modification.

TABLE 2
Description of the Transitions

| Executor | Transition | Description |
|---|---|---|
| User | start | The user enacts the activity from the to-do list. |
| | suspend | The user temporarily suspends the work. |
| | submit | The user submits the completed activity. |
| | create | The user adds a new activity instance. |
| | modify | The user modifies an existing activity instance. |
| | delete | The user deletes an activity instance. |
| System | enable | An activity gets enabled after the completion of one of its predecessors. |
| | disable | An activity gets disabled after a choice at a branch routing node. |
| | block | The activity gets blocked when a process designer enters the modification mode. |
| | free | The activity gets ready again when the user leaves the modification mode. |

adopted for the process definition phase. The "correct-by-construction" approach can be assessed by considering the operational semantics of the operations that may alter the structure of a process definition:

- *Inserting* a new modeling construct *after* an existing one. The respective operation is defined as $insertAfter(C, C_{pre})$, with $C$ being the new construct and $C_{pre}$ being the construct after which $C$ is to be inserted. Fig. 3 shows the situations that may occur: 1) a new construct may be inserted after an existing construct (Fig. 3a) or 2) a new construct may be inserted after a split node, thus causing the creation of a new branch in the respective step (Fig. 3b).
- *Inserting* a new modeling construct *before* an existing one. The respective operation is defined as $insertBefore(C, C_{succ})$, with $C$ being the new construct and $C_{succ}$ being the construct before which $C$ is to be inserted. Fig. 4a shows how to insert a new construct before an existing construct, Fig. 4b shows how to insert a new construct before the first construct in a branch of a step (the same approach also applies to the body of a cycle).[6]
- *Defining a step* starting from an already existing concatenation of constructs, which must be a structured process as defined in Section 2.2. The existing concatenation thus becomes a branch of the step, while other branches are to be added afterward. Fig. 5 graphically illustrates the modification. The respective operation is defined as $step(type, C_{first}, C_{last})$, with $type$ being either *Or-step* or *And-step*, $C_{first}$ being the first construct of the concatenated constructs, and $C_{last}$ being the last construct of the concatenated constructs. As side effect, the creation of a step causes the insertion of a split node *before* $C_{first}$ and of a join node *after* $C_{last}$, as well as the creation of the construct $C_3$ to be further specified, according to the general structure of the step construct (cf. Fig. 1).
- *Defining a cycle* around an already existing concatenation of constructs, which must be a structured process as defined in Section 2.2. The existing concatenation thus becomes the body of the cycle (cf. Fig. 5 that shows the creation of a step; the creation of a cycle proceeds analogously, but without the need to define additional branches). The respective operation is defined as $cycle(C_{first}, C_{last})$, with $C_{first}$ and $C_{last}$ being the first and the last construct of the concatenation, respectively. This causes the insertion of a LoopJoin *before* $C_{first}$ and of a LoopSplit *after* $C_{last}$.
- *Deleting* a modeling construct, by using the operation $delete(C)$, with $C$ being the construct to be deleted. Deleting an existing construct may have an intuitive meaning like in Fig. 6a, or it may present side effects, as in Figs. 6b and 6c. Indeed, if one deletes the last construct of a branch in a step (Fig. 6b, first step), the whole branch is removed from the step; if the last construct of the last branch is removed, the whole step (i.e., its split node, the construct, and the join node) is deleted (Fig. 6b, second step). If the last construct of a cycle is removed, the whole cycle construct is deleted (Fig. 6c).
- *Modifying* an activity, by using the operation $modify(A)$, with $A$ being the activity to be modified. Modifying properties and associations of an activity

6. The distinction between insertions *before* and *after* is required to allow the addition of new constructs in each possible position in a process definition and the creation of new branches in steps.

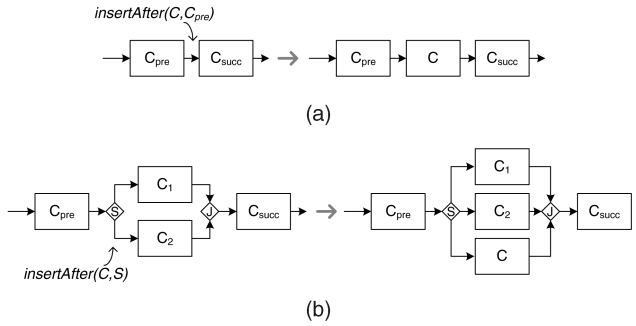



Fig. 3. *insertAfter* operation. (a) Creation of a new modeling construct *after* another construct. (b) Creation of a new branch in an existing step.

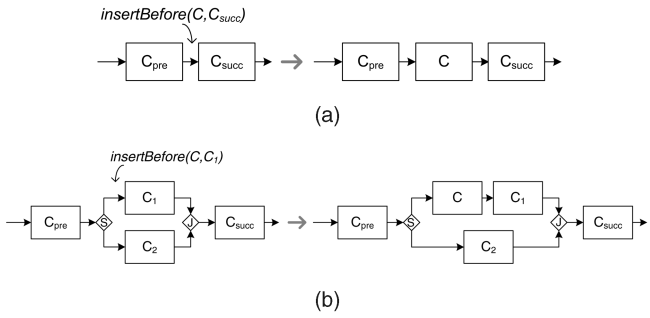



Fig. 4. *insertBefore* operation. (a) Creation of a new modeling construct *before* another construct. (b) Creation of a new construct in an existing branch.

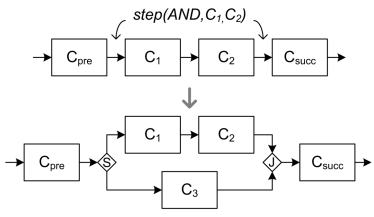



Fig. 5. Defining a new step construct starting from existing constructs.

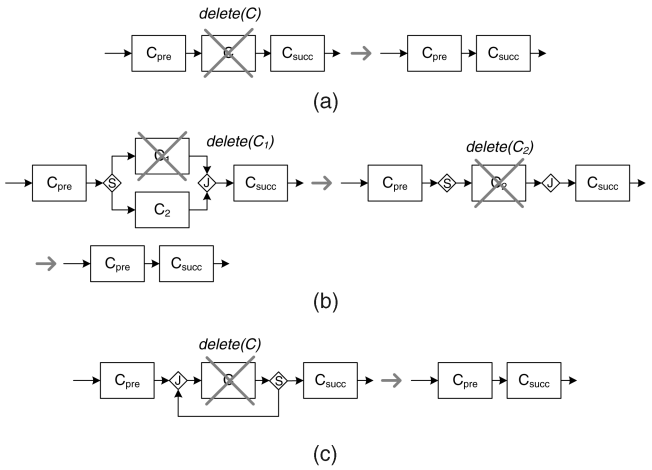



Fig. 6. Delete operation. (a) Deletion of a modeling construct. (b) Deletion of the last construct in a branch. (c) Deletion of the last construct in a cycle.

with resources or users does not alter the process structure and, hence, does not represent a threat to the correctness of the process. Modifying an existing activity grants the process designer access to the activity's configuration panel, already used for the instantiation of the activity. There is no modification operation for steps or cycles.

Considering that the process model underlying our flexible processes is well formed and given the previous characterization of the possible operations that may be performed on a flexible process model, we can say that

**Theorem 2.** *Flexible process models are always* well formed.

**Proof.** Modification operations are applied to structured, well-formed process models. None of the legal modification operations breaks the structuredness of the input process:

- The insertion of a *new construct* before or after an existing construct corresponds to a concatenation (cf. Theorem 1).
- The creation of a *new branch* in an existing step is performed in accordance with the construction rule of the step construct (cf. Section 2.2).
- A *new step* or a *new cycle* may only be defined around structured constructs. Therefore, there are no spare splits or joins (splits or joins without corresponding join or, respectively, split) in the body of the step or cycle.
- The *deletion* of an existing construct takes into account the structure of the process model (cf. Fig. 6).
- The *modification* of the properties of an activity does not impact on the structure of the process model.

Hence, the allowed operations preserve the well formedness of the input process, i.e., they never lead to an AndSplit followed by an OrJoin or to an OrSplit followed by an AndJoin. □

We can conclude that, just as the high-level guidance during the definition of a new process allows us to ensure the *correct construction* of the process model (see Section 2.2), the semantics of the create, modify, and delete operations allows us to guarantee the *correct modification* of a process model during runtime.

### 2.5 Process Templates

Templates are predefined processes, available to users; they can be considered as a library of predefined processes, available to them. Users copy templates into processes, which are then identical to any other process created from scratch; however, some of the activities of a template can be tagged as *mandatory*.

**Definition 8 (Process template).** *A* process template *is a process model that includes the possibility of tagging some of the activities as* mandatory.

Mandatory activities must be executed in any legal enactment of the process (hence, they cannot be directly or recursively included within disjunctive steps). When a template is copied into a process, the process inherits these constraints; therefore, mandatory template activities cannot be removed from such processes, neither due to their deletions, nor due to the creation of steps such that the mandatory activity can be bypassed. Moreover, if mandatory activities are related by a precedence relationship, then that precedence relationship must be preserved when the process is modified.

Setting an activity as mandatory is a choice of the template designer, who can constrain the way in which all processes derived from the template may evolve. For instance, the activity of delivering a final report could considered mandatory in a template that otherwise includes arbitrary activities. That same template could include as initial mandatory activity the definition of requirements, and then any derived process would include the activities of requirement definition and of final report delivery, with the former one occurring earlier than the latter one. In light of these considerations, the use of template-based processes can be considered as an instrument for the specification of *process invariants*, i.e., of a set a minimum requirements that process definitions must satisfy.

## 3 Flexible Processes in E-Learning

Computer Supported Collaborative Learning (CSCL) provides learning environments where learners' teams collaborate by means of computer-mediated services with the aim of reaching a common learning goal [15]. In this context, learning environments are organized so that students can 1) act individually, by producing separate results later combined to achieve a group result, 2) cooperate, by sharing and discussing ideas, and 3) jointly collaborate on some artifacts, possibly following some planned procedures, to reach a team result.

Collaboration requires the most intensive form of interaction, since it also implies some form of coordination [14] based on the definition of processes guiding the learners' activities. The need to support coordination through process-based learning is becoming more and more relevant: some studies have indeed showed that enabling learners to plan their collaboration, by allowing them to define processes on their own, is a success factor to improve learners' productivity [9].

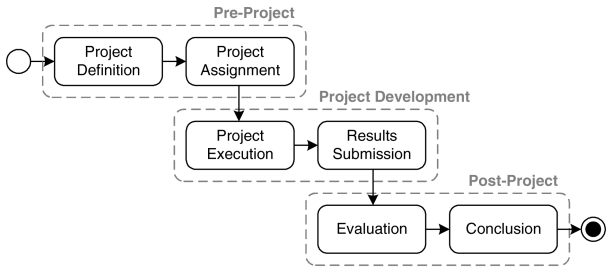



Fig. 7. Phases and activities in the project lifecycle.

In line with the previous observations, the *COOPER* project [6] aims to support team-based, project-centered learning processes, where learners belonging to distributed teams are asked to work on projects and to cooperate to produce some results. In particular, the scenario in which the project moves is shaped up around the lifecycle of projects. As illustrated in Fig. 7, three main phases allow managing and developing projects. In a *preproject phase*, projects are first defined, by evaluating project proposals, which are then assigned to teams. During the following *project development phase*, teams work on the assigned project to achieve predefined goals. Finally, the *postproject phase* addresses the evaluation of results and their dissemination. Dissemination also implies storing results persistently into a knowledge repository describing the full history of the project, available to the user organizations and/or stakeholders.

In this general scenario, our framework for flexible processes has been adopted to support the project development phase, offering facilities for a flexible schedule and organization of collaborative tasks. The result has been the production of a Web-based collaborative platform, covering the requirements for project-based education posed by the users' institutions involved in the *COOPER* project. The platform provides users at work on projects with easily customizable project management activities (e.g., for building teams, for assigning roles and privileges to individuals, for defining work plans and agendas, for agreeing upon decisions, for organizing repositories of documents, etc.), as well as cooperation tools, including asynchronous ones (repository management, forum, and so on), as well as synchronous communication tools (chats or conferencing environments, using Voice-over-IP (VoIP) technology) [1]. In accordance with the process model defined in Section 2, such tasks can also be easily composed into flexible processes, to guide the team activity during the development of their projects.

The rest of this section concentrates on the features of our collaborative platform that supports flexible processes.

### 3.1 The Actors

In order to clarify the organization and functionality of the platform, it is useful to clarify the different roles that users can play in the definition and the execution of flexible processes. The user roles depend on the expertise required to set up the domain- or application-specific activities (work tasks or learning assignments), to organize work into processes, to perform work or learning tasks, and to monitor the executed tasks. The *COOPER* platform reflects these four competencies in the form of four different roles, i.e., *activity designers*, *process designers*, *users* (students), and *tutors* (professors). Typically, the *activity designer* specifies the activity types that are available in the platform, the *process designer* then defines process templates, the *users* perform the actual work or learning tasks, and the *tutor* monitors the process and the users' learning progression and may also define process templates. More precisely, the four roles are defined as follows:

The *activity designer* identifies and defines application-specific activity types, together with their properties and constraints. In this way, she/he tailors the support for flexible processes to the chosen application domain, by extending the activity type library. Activity design typically requires the analysis, prior to the deployment of the platform, of the particular application domain that the platform must address, and therefore the activity designer operates at design time. Once the activity type library has been defined, and thus the platform has been set up, two further roles then take part in the definition and execution of processes.

Starting from the activity type library and/or from predefined templates, the *process designer* instantiates activities and composes them into processes. In most cases, the activity designer is a *tutor* supervising the team during the project development who wants to schedule and organize the team activity. More specifically, it may happen that the tutor defines process models in form of templates, to suggest a basic organization for the team collaboration. Team members can also play the role of process designers. They are indeed enabled to define new processes by extending templates, or by composing new models from scratch.

Given a process definition, the *users* are then the actors who have been assigned with some activities within the process, and that therefore perform such activities during the process execution. Users are also allowed to modify process definitions during runtime, after the process has been launched, with the only limit of not violating the template constraints that the process may hold by definition.

A relevant feature in e-learning scenarios is assessment. At this proposal, *tutors* are allowed to monitor process execution; they have, therefore, access to monitoring interfaces that enable them to evaluate the activity of the whole team as well as of each individual user.

### 3.2 The E-Learning Activity Type Library

Based on the e-learning scenarios analyzed within the *COOPER* project, we have developed a library that includes some 40 activity types, classified according to the main cooperation goals they are related to the following:

- *Teamwork planning.* This category refers to the organization and the scheduling of the team work, which is supported by activity types such as: "Assign roles," "Collect team member competencies," "Define tasks," "Assign tasks," "Agree on task division," "Define milestones," and "Plan deliverables."
- *Resource management.* This category refers to the tasks for publishing, accessing or also recommending resources (e.g., documents, forum messages, wikies, etc.). The related activity types are "Publish re-

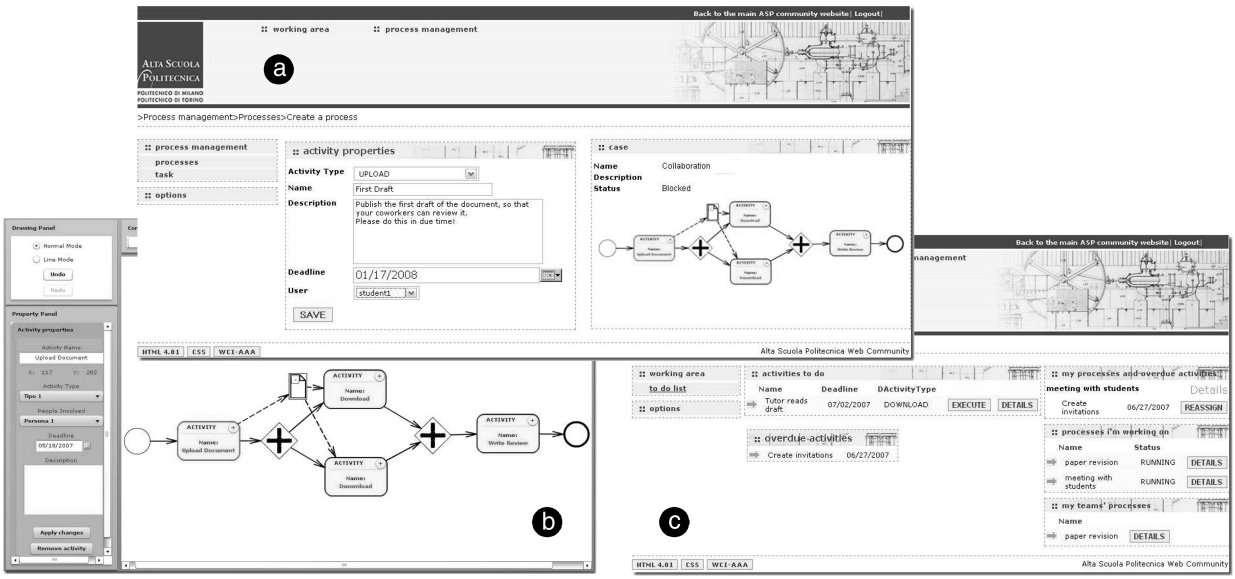



Fig. 8. Screen shots of the *COOPER* platform. (a) Form-based process definition with visual preview. (b) Editor for visual process definition. (c) The user's to-do list.

sources," "Acquire resources," and "Recommend resources."

- *Communication.* This category refers to tasks for the invocation of synchronous [1] and asynchronous communication services. The activity types in this category include making a VoIP call, creating, opening, moderating, joining, and closing synchronous activities (such as a one-to-many videoconference), moderating meetings, defining and taking part to chat rooms, co-browsing or co-editing for application sharing, and voting through polls.
- *Reviewing and assessing.* This category covers some reviewing activities, as well as assessment for team members, for themselves and also in the context of the project team [30]. Activity types in this category are "Creating Review Reports," "Designating reviewers," "Submitting reviews," and also "Define assessment criteria," "Define performance indicators," and "Plan assessment."

For all the previous activity types suitable hypertext interfaces have been defined, which allow users to execute the activities instantiated starting from such types.

### 3.3 Web-Based Definition and Execution of Processes

Our collaborative platform makes use of a Web front end, which provides users with easy-to-use interfaces for the definition and execution of their collaboration processes.

Process definition is performed by means of a form-based and/or a visual process editor (see Fig. 8a), which do not require users to be aware of process modeling concepts. Users are indeed "guided" to apply the composition rules illustrated in Section 2; the result is the construction of a structured process that can then be executed correctly.

Just to give an example of process definition, Fig. 8a reports a page from our process editor, which allows the user to instantiate an activity within a process, starting from an activity type. The user selects the activity type (e.g., "UPLOAD"), and then enters the activity name and the values for all its properties, e.g., the activity description and the deadline for the activity execution. She/he then assigns the activity to an individual user or to a group of users (to instantiate the *act* function in the process model). Depending on the activity type, the user may also associate the activity with some resources to manage possible document flows (the *res* function in the process model). The process definition can then proceed by adding other modeling constructs (activities, steps, or cycles), as allowed by the construction rules illustrated in Section 2 until the end of the process is reached. In particular

- as illustrated by the example above, if the process designer adds an activity, she/he is required to specify the activity properties, the associated users and possible resources; the system then automatically creates the connection with its previous constructs (corresponding to the *succ* function in the process model);
- if a process designer adds an And-step or an Or-step, the system automatically creates the required routing nodes (split and merge) and also defines the required connections of the routing nodes with the previous and next constructs (the designer is just required to specify the branches of the step); and
- if a process designer adds a cycle, the system automatically creates a LoopJoin and a LoopSplit, and all their required connections with the constructs preceding and succeeding the cycle (the designer is then required to specify the constructs forming the cycle body, plus the condition that will control iterations during process execution).

As showed in Fig. 8a, process creation by means of the form-based paradigm is also accompanied by a visual representation of the process diagram under construction.

In each moment, users are able to shift to the visual paradigm, which leads them into a page (see Fig. 8b) where they can directly compose the graph in a WYSIWYG manner.

If the process definition is performed through the form-based paradigm, the process model is constructed from left to right. For example, steps are constructed by specifying a branch at a time, with a "depth-first" procedure: the definition of a branch must be completed before the definition of another parallel branch in the same step can start. The "left to right" construction is not preserved when the visual modality is adopted for process definition, since the user is potentially allowed to visually insert constructs in any order; but the definition procedure still reflects the construction rules illustrated in Section 2. For example, the addition of a step implies the automatic insertion of the corresponding routing nodes. Also, some validation rules allow checking the correctness of constructs, as the designer inserts them into the diagram as well as when the diagram is finalized. These features guarantee that the resulting process definitions are well formed.

Once a defined process is launched for execution, the involved users are notified through their *to-do* list. Fig. 8c depicts the page where a user can see the state of her/his tasks, having access to the list of the processes she/he is involved in, with the possibility to get an overview of the process activities involving the whole team, as well as details about her/his activities. The user is also alerted about her/his overdue activities. She/he can therefore start the execution of her/his pending activities. Every time an activity is started, the user is redirected to the page(s) enabling its execution.

During process execution, tutors in charge of supervising teams are then provided with a monitoring panel that summarizes the execution state of the process activities for all their teams. Tutors can access the details of all the activities. They can also filter the completed activities as well as the overdue activities. For the latter, they can undertake some "compensation actions," such as assigning them to other users or extending the deadline.

## 4 Implementation

Fig. 9 illustrates how the application features described in the previous section are composed into one comprehensive Web application, i.e., the overall *COOPER* platform. Starting from the discussed Web front ends of the *COOPER* environment, the functional architecture depicted in Fig. 9 shows how the single modules interoperate and make use of data and metadata.

Process definition is performed via a *process editor*, which makes use of the predefined *activity type library* and, possibly, of existing *process/template models.* Process execution is performed via the *COOPER*'s *collaboration environment*, which leverages on the hypertext front ends of the predefined activity types to allow users to produce and consume process data in form of resources stored in the *resource repository.* Process advancement is governed by the stored process definitions, which are interpreted during process execution by a dedicated *process engine* that contains the necessary application logic to maintain the running

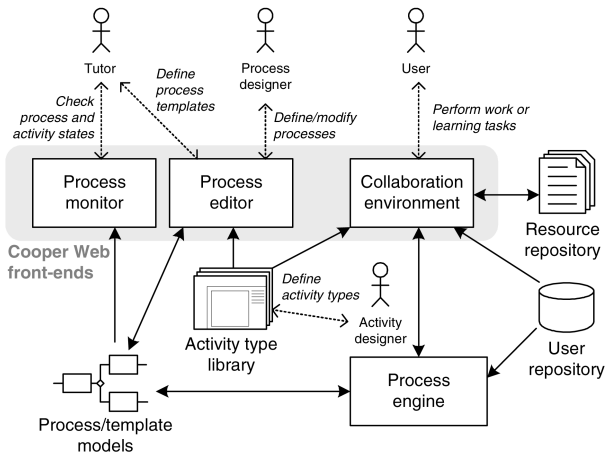



Fig. 9. The conceptual architecture of the *COOPER* framework, supporting the definition and execution of flexible processes.

processes' metadata and, hence, to drive the activity flow in the collaboration environment. During the execution of a process, it is possible to check the status of the process and of the single activities composing the process by means of the *process monitor.* Fig. 9 also highlights the competences of the individual actors in the *COOPER* platform.

The design and implementation of the above Web platform for project-centered e-learning has been achieved by suitably extending Web Modeling Language (WebML [12]), a conceptual modeling language and development method for data-intensive Web applications. The choice of WebML for the implementation of the platform is justified by both its data-driven and model-driven approach. The data-driven approach fits best to the need for a simple and efficient management of process data and metadata and to the need for easily sharing such process data and metadata among different application components, i.e., collaboration environment, process editor, and process monitor. The model-driven design fits best to the need for a fast and efficient development process, as the one enabled by the automatic code generation technique that accompanies the WebML method [33].

The main extensions to the modeling language that have been developed in the context of the *COOPER* project refer to novel, collaboration-specific requirements in Web application design such as[7]

- *instant messaging* for synchronous text-base discussions,
- *push & speak* for VoIP telephony,
- *audio/videoconferencing tool* for VoIP phone conferences among a group of people, also equipped with webcam-based videoconferencing support, and
- *application sharing* for the collaborative, synchronous work on one shared application, e.g., by means of the novel *co-browsing* feature.

The previous features represent important enabling technologies for the synchronous communication among team members. While asynchronous communications (e.g.,

7. For a detailed description of the *COOPER* extensions for WebML, refer to [1].

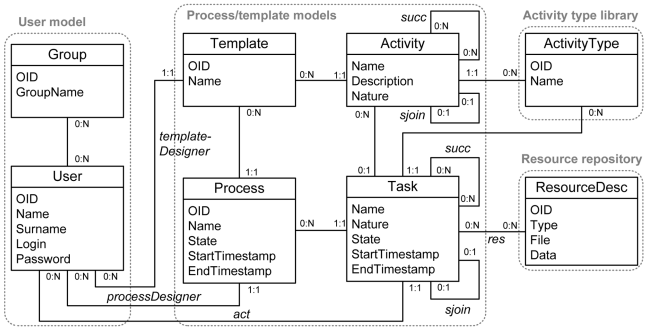



Fig. 10. The data model of the *COOPER* platform.

email and discussion forums) could easily be implemented by means of the standard WebML constructs, the described synchronous communication mechanisms have required a proper extension of the language and the underlying runtime framework.

According to the WebML development process, the implementation of the *COOPER* platform has therefore proceeded in two main development steps: data design and hypertext design.

### 4.1 Data Design for Flexible Processes

The adoption of a data-driven paradigm for the management of flexible processes requires that different elements describing process *data* (resources and documents created or consumed by users) and process *metadata* (process model, runtime process management data) be explicitly represented as data in the data layer underlying the *COOPER* platform. Accordingly, the process model for flexible processes introduced in Section 2 has been formalized by means of appropriate data entities and relationships that, during process definition and execution, maintain the necessary runtime data.

The result of this formalization is summarized in Fig. 10. The dashed boxes in the Entity-Relationship diagram show how the single components of the process model contribute to the four information repositories identified in Fig. 9. At the left-hand side, the *user model* contains the basic identification and authentication data (entity `User`), as well as the entity `Group`, which provides for the association of users with user groups and, thus, for the management of access rights. The relationships `templateDesigner` and `processDesigner` also express the association of some users playing the role of template designer or process designers with the corresponding templates and processes.

At the top right corner of the figure, we have the *activity type library*, which in its simplest version only contains the name of the hypertext portion providing the Web interface for the execution of the activity. As we will see in the following, the hypertext portion corresponds to an *area* in the WebML hypertext schema. In the lower right corner of the figure, we have the *resource repository*, containing the descriptions and references to the data objects produced or consumed during process execution, typically attachments in form of files stored in the platform. In the center of the figure, we finally have the *process/template models*, which represent the main process metadata required for the execution of the flexible processes. In the top part, the entities `Template` and `Activity` store template models, in the lower part, the entities `Process` and `Task` store process models. Template models cannot be executed directly and only serve to instantiate process models, which is achieved by copying all template-specific metadata from the entities `Template` and `Activity` to the entities `Process` and `Task`. Process models, on the other hand, may also be executed; the entities `Process` and `Task` therefore also contain some runtime data required for the automatic management of the process advancement and the monitoring of the process state. The relationship between the entity `Task` and the entity `ActivityType` is required to track the association of tasks that are created after the instantiation of the template with their activity type; the activity types of the tasks that are part of the original template definition are inherited from the entity `Activity` at process instantiation time.

As for the mapping of the process model introduced in Definition 6, we can thus say: The set $A$ of activities is represented by the entity `Task`, the set $R$ of resources is represented by the entity `ResourceDesc`, and the set $U$ of users is represented by the entity `User`. The functions $succ$, $act$, and $res$, instead, are modeled by means of the relationships `succ`, `act`, and `res`. The relationship `succ` assigns to each activity of a process a set of successor activities; the relationship `act` assigns to each activity a user; and the relationship `res` assigns to each activity the possibly used resources.

The last named relationship in Fig. 10 is `sjoin`, which is required to keep track of the start and end system activities of the modeling constructs *step* and *cycle.* More precisely, this is achieved by assigning to the step's or the cycle's opening system activity (*AndSplit*, *OrSplit*, or *LoopJoin*) its respective closing system activity (*AndJoin*, *OrJoin*, or *LoopSplit*). This technique is necessary to guarantee the correctness of the process models, as described in Sections 2.2 and 2.4.

It is worth noting that, due to the adoption of self-described processes, the entities `Process` and `Task` are used both at design time, to store the initial process model, and at execution time, to capture possible process modifications and runtime metadata for process monitoring. The management of these data is performed by the *process engine*.

### 4.2 Hypertext Design

The model-driven, visual paradigm of WebML for the specification of the application's hypertext front ends is tightly coupled with the organization of the application's data. The previously described formalization of the process model as application data (Fig. 10), therefore, represents the starting point for the development of the Web front ends of the *COOPER* platform. In line with the focus of this paper, we limit the following discussion of the development of the hypertext front ends to the part concerning the execution of flexible processes in the *COOPER* platform; for the complete discussion of the platform and its implementation, the reader is deferred to [6] and [13].

Fig. 11 shows a simplified WebML hypertext schema that shows how, during process execution, the *COOPER* platform dynamically computes the process flow that needs to be followed for each user. Starting point for this computa-

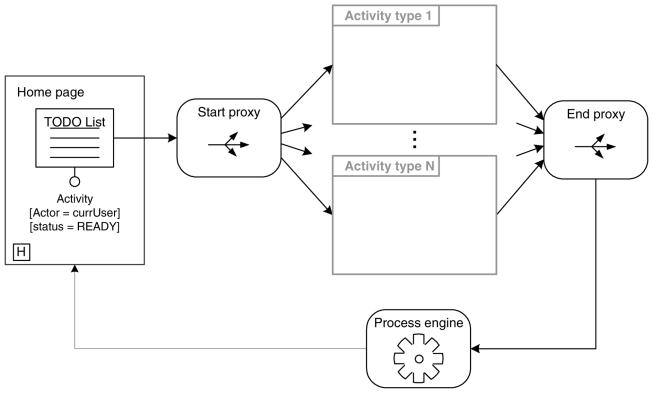



Fig. 11. Simplified WebML hypertext schema that shows how the *activity type library* (the set of WebML areas that is hinted at in the center part of the figure) is connected to the overall *COOPER* platform.

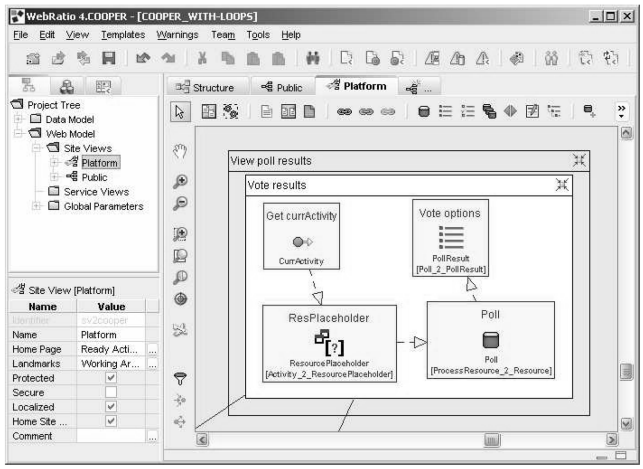



Fig. 12. The WebML hypertext schema of the *View poll results* activity type in the WebRatio CASE tool.

tion is the activity type library introduced in Section 2 and represented in Fig. 11 by means of a set of WebML *areas* (for presentation purposes, the areas in the figure are left empty), which contain for each activity type the hypertext portion that represents the actual interface that is presented to the user for the execution of the activity.

Hypertext computation during process execution therefore proceeds as follows: A user accesses her/his `Home page` (the rectangle labeled with an `H` in the left part of the figure) by logging in to the *COOPER* platform. The platform provides the user with her/his personalized list of *ready* activities she/he can choose from. Selecting one of the activities invokes the `Start proxy` unit (representing a business logic operation in WebML), which selects the appropriate hypertext portion associated with the activity chosen by the user. Choosing a hypertext portion means forwarding the user to the start page of the respective area in the hypertext schema. Fig. 12, for example, shows how a hypertext portion of an activity type is defined internally (in this case, the WebML model is showed as it is displayed in the WebRatio CASE tool): Just as the rest of the platform, an activity's hypertext specification makes use of *pages* (one page in the case of the `View poll results` activity depicted in Fig. 12), *content units* that are rendered inside the hypertext pages, and possibly *operation units* that are placed outside the pages to model the business logic of the activity type. Each activity is completed by means of a *Complete* command, which connects the hypertext portion of the activity to the `End proxy` unit (at the right-hand side in Fig. 11), which forwards the hypertext computation to the `Process engine`.[8] The engine is in charge of keeping up-to-date the process metadata required for the management of the process executions. For example, the process engine manages the activity states and transitions described in Fig. 2. Finally, after completion of an activity, the user is presented with an updated `Home page`, showing the new list of ready activities.

The model-driven approach highly facilitates the extension of the activity type library with new activity types. The addition of a new type in fact only requires the modeling of the respective hypertext area (similarly to the one illustrated in Fig. 12) in the visual WebML modeling tool, the generation of the corresponding application code, and the deployment of the activity in the *COOPER* platform. In short, adding a new activity type requires the introduction of new Web pages into the platform, without being forced to know how the rest of the platform is implemented and interconnected. This feature directly derives from the fact that process execution is totally independent of the activity types actually available in the platform and largely facilitates the extension and the customization of the platform to the needs of varying institutions and application domains.

### 4.3 Implementation and Deployment

As already hinted at in Fig. 12, the availability of the *COOPER*-specific communication extensions for WebML and the novel support for flexible processes has allowed us to develop the entire platform with the help of the WebML CASE tool, WebRatio [33], in a fully WebML-conform fashion. With the help of WebRatio's code generators, the entire platform has been automatically generated on top of a J2EE platform and deployed in a Tomcat Web server. The final WebML project of the platform spans several hundreds of WebML elements, comprising pages, content and operation units, links, and areas.

### 4.4 User Experience

The design of the collaboration environment described in this paper descends from the requirement analysis conducted together with two academic institutions involved in the *COOPER* project [6]:

- Advanced Learning and Research Institute (ALaRI—http://www.alari.ch), a school offering master programs at the Università della Svizzera Italiana.
- Alta Scuola Politecnica (ASP—http://www.asp-poli.it), a school for talented students founded by Politecnico di Milano and Politecnico di Torino.

The learning activities at ALaRI and ASP are organized around geographically distributed teams working on projects. In particular, teams need to create their own work plan and use communication tools. A prototype of our collaborative platform was deployed at both institutions; we then evaluated at ASP the use of flexible processes and

8. In Fig. 11, the process engine is symbolically represented as WebML operation unit, but the actual business logic of the engine spans several dozens of operations, which have been omitted for presentation purposes.

VoIP communication. We enrolled five different teams, for a total of 25 students. The evaluation lasted for 6 months (from April 2007 to October 2007) and was organized into two steps: 1) *preproject evaluation*, aimed at evaluating the students' expectations and 2) *postproject analysis* of the students' feedback and of the system logs, aimed at evaluating the real usage of the platform [26]. To support the evaluation, we extended our platform with a module that manages questionnaires by offering functions for publishing questions, collecting users' answers, and automatically generating reports [26].

Before starting the evaluation, two training seminars were organized to illustrate the platform organization; the majority of students (80 percent) did not need further training. Then, students were asked to fill in an online questionnaire with 25 closed questions, addressing their expectations on the team collaboration improvements that could be achieved from the use of flexible processes and their preferences about communication tools. Their answers indicated that

- Fifty-four percent of students expected the platform to be useful for improving the effectiveness of communication between them and their teams and teachers,
- Eighty-three percent of them expected to spend less time for the development of their projects, due to the positive effect induced by the coordination of team activities, and
- Sixty-five percent of them expressed a clear preference for VoIP tools (used for communication in our platform), while 35 percent of them prefer other channels, such as chats and forums.

After this preliminary evaluation, students used the platform for six months, for coordinating their team work by means of synchronous communication tools and flexible processes. Logs of users' interactions were collected.

A postproject questionnaire was submitted to the students after the evaluation period, to collect their feedback about the ease of use of the interface, the effectiveness of the approach, and the overall satisfaction. Some questions evaluated the communication activities described in Section 3.2, as they are crucial for the usability of a collaborative platform. Users were asked to give marks from 1 (very bad) to 4 (very good) to different aspects of the platform:

- Seventy-four percent of the students rated the interface as "easy to use" (40 percent of the respondents scored the item with 4 out of 4).
- Eighty percent of them rated the platform interface as "quite effective" (a score of 3 or above out of 4). The average score given to the overall satisfaction in the use of the platform is 2.8, an indicator of the users' general positive perception of the platform usability and effectiveness.
- Seventy-five percent of the participants positively rated the use of the synchronous tools (chat and VoIP), outperforming the result of the pre project questionnaire, while 65 percent gave a positive opinion on asynchronous communication tools (specifically, a forum).

The usage logs show that, after a few weeks of acquaintance to flexible processes, students managed to adapt the system to the needs of their teams. They were indeed able to define and modify processes so that their goals could be achieved in time. In fact, the rate of uncompleted or reassigned activities decreased, while the processes sketched in preliminary phases of the evaluation were successfully reshaped by means of the operations described in Section 2.4, to better suit the team's way of working.

## 5 Related Works

Given the always increasing emphasis on the "learning-by-doing" paradigm, several research and industrial efforts are being devoted to enhance e-learning teamwork activities. Approaches range between two broad classes, *groupware* and *WfMSs*.

Communication support systems or groupware (such as LearningSpace, [17], Lotus Notes and Domino [21], MS Groove [23], BSCW [24], Blackboard and WebCT [5], etc.) offer facilities for workspace creation, resource sharing, and synchronous and asynchronous communication, especially aimed to support rapidly changing, nonrepetitive processes. However, while some studies have demonstrated that supporting learning by structured collaboration processes is a success factor for the team productivity [9], the majority of the proposed systems are still "task-oriented," not "process-oriented" [20]. More specifically

- they support individual tasks, offering very limited support to sustain collaborative processes,
- they provide a given set of hardwired collaboration tasks, and their extension toward unanticipated collaboration activities is difficult or even impossible, and
- even when they offer facilities for collaborative process definition, they lack a high-level, global perspective over the defined processes, as they offer a low-level interface based on parameter setting and component composition within scripts.

As a consequence, with such platforms, users have difficulties in understanding the ongoing processes and keeping track of the accomplished tasks.

On the opposite site, WfMSs (see [32]) are too rigid to support the variable nature of collaborative processes. They indeed ask application designers to predict at design time the structure of the processes; then the runtime modification of processes and the addition of new tasks become cumbersome.

Positioned between the two extremes are *evolving workflows* [11], [29], which support partially specified processes, i.e., processes where only part of the flow can be adequately predefined, while the full specification of the activities and their sequencing can be completed during process enactment. In [28], authors identify three dimensions characterizing workflow evolution:

- The first dimension is *dynamism*, which is based on the assumption that the predefined models underlying processes have to be changed if the related business processes change [18], [14]. The more severe problem emerging in this context is the compliance of the already active workflow instances,

enacted on the basis of the original model, with the new process specification [11].

- The second dimension is *adaptability*, which is the ability of a process to react to exceptional situations. The majority of exceptions that could occur at execution time can be anticipated when designing the process [10]; however, the complete set of exceptions for a given process can never be captured, and dealing with not planned exceptions requires several efforts [27].
- The third dimension is *flexibility*: the process is executed on the basis of a loosely (or partially) specified model, and the full definition of the process is provided at runtime, and may be unique to each instance.

Our approach focuses on flexibility. In the literature, very often flexibility has been managed by providing extensions to workflow definition languages, for example, by introducing new and more complex control constructs supporting the variability of the activities to be enacted at runtime and/or their sequencing [31]. However, the new languages often result into proprietary solutions requiring complex implementations of execution engines, thus lacking portability.

Very few proposals have so far emerged that, in line with our approach, support flexible processes in virtual teams (see, for example, [16], [4], and [14]). They provide users with environments for process definition and modification. More specifically, some proposals address "flexible e-learning" and introduce environments based on workflow technologies, where users can define their own learning paths and collaboration is driven by flexible, yet controlled, means of progressing through the defined processes. However, very often in such proposals flexibility covers limited features. For example, in the Flex-el learning environment [20], flexibility is limited to the relaxation of time constraints. Also, in [29], authors provide support for *pockets of flexibility*, i.e., special build activities, embedded in the process model, that allow users to define the sequencing of some workflow fragments predefined in a process template.

Our notion of flexibility is broader. We enable users to fully play the role of process designers, by allowing them to modify at runtime predefined templates (updating activity properties and sequencing, as well as adding and deleting activities), and also to define completely new processes in order to meet the coordination and collaboration requirements of their teams. Such a level of flexibility is achieved thanks to our notion of *activity types*, which is the basis for the runtime composition and modification of processes. This is in line with the recent proposal of the *Activity-Based Computing* (ABC) paradigm [3], which aims at supporting the organization and execution of parallel users' activities, distributed in time and space. According to this paradigm, the basic computational unit is no longer the file (e.g., a document) or the application (e.g., a word editor), but the activity of a user. The user is thus enabled to initiate, suspend, store, and resume activities. ABC also supports collaboration through asynchronous and synchronous activity sharing.

ABC is inspired to the Activity Theory [22], a framework focused on the conceptualization of the dynamic and situated nature of workflows [2]. The Activity Theory paradigm has also been proposed for the description of collaborative learning processes [8]. However, all the approaches based on it still focus on predefined and static processes, without supporting the definition of collaboration processes by users. We instead believe that supporting process definitions—through user-friendly interfaces—is essential for enhancing teamwork activities.

## 6 Conclusions

In this paper, we have presented a framework for the design and execution of flexible collaborative processes. The solution has been conceived in the context of project-centered learning, a scenario experienced by two academic institutions in the context of an EU-funded project. However, the characterization of our approach is domain-independent and can be used in a variety of other contexts—provided that the typical activities for that context are detected, specified, and implemented as Web applications. As the flexible process model resulting from the described research is general in nature, and an implementation of the respective runtime support in form of a specific extension of an existing e-learning tool (e.g., Moodle) would have limited its applicability, we have opted for an extension of WebML and WebRatio. This choice will allow us to easily apply the conceived solutions to other collaborative domains and to further assess the model's viability. The experiments in the e-learning domain have already demonstrated the usefulness of flexible process support and the appreciation of the proposed environment by students.

Our environment is already fully functioning. Our current activity focuses on extensively experimenting the framework to improve its effectiveness and efficiency with respect to the user needs. We are also working on extending the WebRatio development environment, by means of a "wizard" to further facilitate the definition and extension of the library of activity types, and by developing effective and easy-to-use monitoring applications.

A promising direction for future work is the identification of process templates by mining process execution data. Given that activities are well understood (they belong to known types), the mining of the process execution is trivial, and reconstructing templates as the recurrent processes performed by users is possible. This mining activity can be regarded as a bottom-up process design method—a quite novel approach, given that process modeling is regarded as prototypical top-down method.

Some further efforts are being devoted to improve the portability of the defined processes, by complementing the data-driven representation of processes with more standard and portable specifications, such as the XML Process Definition Language (XPDL) [34] proposed by the Workflow Management Coalition.

## Acknowledgments

This work is founded by the FP6 EU Project *COOPER* (IST027073). The authors are grateful to Professor Jan van

Bruggen and to Howard Spoelstra for the help offered in the identification of the activity types for the e-learning domain, Vlad Posea for providing the data of the *COOPER* platform evaluation, and the students and staff at ASP and ALaRI.

## REFERENCES

[1] N. Aste, A. Bongio, S. Ceri, M. Fais, M. Matera, and A. Raffio, "Model-Driven Design of VoIP Services for E-Learning," *Proc. First Int'l Workshop Collaborative Open Environments for Project-Centered Learning (COOPER '07),* CEUR Workshop Proc. (CEUR-WS.org), vol. 309, 2007.

[2] M. Adams, E. Edmond, and A.H.M. ter Hofstede, "The Application of Activity Theory to Dynamic Workflow Adaptation Issues," *Proc. Seventh Pacific Asia Conf. Information Systems (PACIS '03),* pp. 1836-1852, 2003.

[3] J. Bardram, J. Bunde-Pedersen, and M. Soegaard, "Support for Activity-Based Computing in a Personal Computing Operating System," *Proc. SIGCHI Conf. Human Factors in Computing Systems (CHI '06),* pp. 211-220, 2006.

[4] P. Barthelmess and C.A. Ellis, "The Neem Platform: An Evolvable Framework for Perceptual Collaborative Applications," *J. Intelligent Information Systems,* vol. 25, no. 2, pp. 207-240, 2005.

[5] Blackboard Academic Suite, Blackboard, http://www.blackboard.com, 2007.

[6] A. Bongio, J. van Bruggen, S. Ceri, V. Cristea, P. Dolog, A. Hoffmann, M. Matera, M. Mura, A. Taddeo, X. Zhou, and L. Zoni, "COOPER: Towards a Collaborative Open Environment of Project-Centred Learning," *Proc. First European Conf. Technology Enhanced Learning (EC-TEL '06),* vol. 4227/2006, pp. 561-566, Oct. 2006.

[7] M. Brambilla, S. Ceri, P. Fraternali, and I. Manolescu, "Process Modeling in Web Applications," *ACM Trans. Software Eng. and Methodologies,* to appear.

[8] M. Caeiro, L. Anido, and M. Llamas, "A Critical Analysis for IMS Learning Design," *Proc. Conf. Computer Support for Collaborative Learning (CSCL '03),* pp. 363-367, 2003.

[9] A. Carell, T. Herrmann, A. Kienle, and N. Menold, "Improving the Coordination of Collaborative Learning with Process Models," *Proc. Conf. Computer Support for Collaborative Learning (CSCL '05),* pp. 18-27, 2005.

[10] F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi, "Specification and Implementation of Exceptions in Workflow Management Systems," *ACM Trans. Database Systems,* vol. 24, no. 3, pp. 405-451, 1999.

[11] F. Casati, S. Ceri, B. Pernici, and G. Pozzi, "Workflow Evolution," *Data and Knowledge Eng.,* vol. 24, no. 3, pp. 211-238, Jan. 1998.

[12] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera, *Designing Data-Intensive Web Applications.* Morgan Kauffmann, 2002.

[13] S. Ceri, M. Matera, A. Raffio, and H. Spoelstra, "Flexible Processes in Project-Centred Learning," *Proc. Second European Conf. Technology Enhanced Learning (EC-TEL '07),* vol. 4753, pp. 463-468.

[14] F. Charoy, A. Guabtni, and M. Valdes Faura, "A Dynamic Workflow Management System for Coordination of Cooperative Activities," *Proc. First Int'l Workshop Dynamic Process Management (DPM '06),* vol. 4103/2006, pp. 205-216, 2006.

[15] P. Dillenbourg, M. Baker, A. Blaye, and C. O'Malley, "The Evolution of Research on Collaborative Learning," *Learning in Humans and Machine: Towards an Interdisciplinary Learning Science,* E. Spada and P. Reiman, eds., pp. 189-211, Elsevier, 1996.

[16] S. Dustdar, "Caramba—A Process-Aware Collaboration System Supporting Ad Hoc and Collaborative Processes in Virtual Teams," *Distributed and Parallel Databases,* special issue on teamware technologies, vol. 15, no. 1, pp. 45-66, Kluwer Academic Publishers, Jan. 2004.

[17] IBM Learning Solutions, IBM, http://www-304.ibm.com/jct03001c/services/learning/ites.wss/zz/en?pageType=page&c=a0001106, 2007.

[18] P.J. Kammer, G.A. Bolcer, R.N. Taylor, A.S. Hitomi, and M. Bergman, "Techniques for Supporting Dynamic and Adaptive Workflow," *Computer Supported Cooperative Work,* vol. 9, no. 3/4, pp. 269-292, 2000.

[19] B. Kiepuszewski, A.H.M. ter Hofstede, and C.J. Bussler, *On Structured Workflow Modelling,* LNCS 1789/2000, Springer Berlin/Heidelberg, ISSN 0302-9743 (Print) 1611-3349 (Online), 2000.

[20] J. Lin, C. Ho, W. Sadiq, and M.W. Orlowska, "Using Workflow Technology to Manage Flexible e-Learning Services," *Educational Technology and Soc.,* vol. 5, no. 4, 2002.

[21] IBM, Lotus Domino, http://www-306.ibm.com/software/lotus/products/domino/, 2007.

[22] A.N. Leont'ev, *Activity, Consciousness, and Personality.* Prentice-Hall, 1978.

[23] Microsoft Office Groove, http://office.microsoft.com/en-us/groove/FX100487641033.aspx, 2007.

[24] OrbiTeam Software, BSCW, http://public.bscw.de/, 2007.

[25] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom, "Object Exchange across Heterogeneous Information Sources," *Proc. 11th Int'l Conf. Data Eng. (ICDE '95),* p. 251, 1995.

[26] V. Posea, S. Trausan-Matu, and V. Cristea, "Online Evaluation of Students' Opinions about the Collaborative Learning System They Use," *Proc. Int'l Conf. Intelligent Computer Comm. and Processing (ICCP),* 2007.

[27] S. Sadiq, "On Capturing Exceptions in Workflow Process Models," *Proc. Fourth Int'l Conf. Business Information Systems (BIS '00),* Apr. 2000.

[28] S.W. Sadiq, M.E. Orlowska, and W. Sadiq, "Specification and Validation of Process Constraints for Flexible Workflows," *Information Systems,* vol. 30, pp. 349-378, 2005.

[29] S.W. Sadiq, W. Sadiq, and M.E. Orlowska, "Pockets of Flexibility in Workflow Specification," *Proc. 20th Int'l Conf. Conceptual Modeling: Conceptual Modeling (ER '01),* pp. 513-526, 2001.

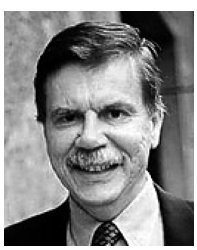
[30] H. Spoelstra, M. Matera, E. Rusman, J. van Bruggen, and R. Koper, "Bridging the Gap between Instructional Design and Double Loop Learning," *Current Developments in Technology-Assisted Education,* A. Mendez-Vilas, A. Solano Marten, J.A. Mesa Gonzalez, J. Mesa Gonzalez, eds., vol. II, Technological Science Education, Collaborative Learning, Knowledge Management, 2006.

[31] W.M.P. van der Aalst and A. Kumar, "A Reference Model for Team-Enabled Workflow Management Systems," *Data and Knowledge Eng.,* vol. 38, pp. 335-363, 2001.

[32] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros, "Workflow Patterns," *Distributed and Parallel Databases,* vol. 14, no. 1, pp. 5-51, 2003.

[33] *WebRatio Site Development Studio,* WebModels s.r.l., http://www.webratio.com, 2005.

[34] *XML Process Definition Language (XPDL),* Workflow Management Coalition, http://www.wfmc.org/standards/xpdl.htm, 2007.



**Stefano Ceri** is a professor in the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy. His research interests include query languages for XML and on models, methods, and tools for the design of data-intensive WEB sites. He is the chairman of LaureaOnLine and the vice-director of the Executive board of Alta Scuola Politecnica. He is responsible for the Politecnico of several EU-funded projects, including COOPER: "Cooperative Open Environment for Project Centered Learning" (2005-2007) and ProLearn "Network of Excellence in Professional Learning" (2005-2008). He is a coinventor of WebML (US Patent 6,591,271, July 2003) and a cofounder of the startup which commercializes WebML by means of the product WebRatio. He is a coeditor in chief of the book series *Data Centric Systems and Applications* (Springer-Verlag). He has authored nine international books and more than 200 articles.

**Florian Daniel** received the master's degree in computer engineering and the PhD degree in information technology from Politecnico di Milano. He is a postdoctoral researcher in the Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento, Povo (TN), Italy. His research interests include business processes, business intelligence applications, conceptual modeling and design of Web applications, adaptivity and context-awareness in Web applications, component-based user interface design, and Web mashups.

**Maristella Matera** is an assistant professor in the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy, where she currently teaches courses on databases. Her research interests include models and design methods for Web applications, adaptivity and context-awareness, and quality analysis of Web applications. She has authored about 100 articles on the previous topics. She is also a coauthor of the book *Designing Data-Intensive Web Applications* (the Morgan Kaufmann Series in Data Management Systems).

**Alessandro Raffio** is a PhD student at the Politecnico di Milano, Milano, Italy, where he is also a teaching assistant in software engineering courses in the Dipartimento di Elettronica e Informazione. His research interests include mapping and transformation languages for structured and semistructured data, as well as Web-based collaboration systems.