We present the GENESIS platform (GEneralised eNvironment for procEsS management in cooperative Software engineering), the outcome of a research project aiming at designing and developing a noninvasive and open-source system to support software engineering processes in a highly distributed environment. The system supports the cooperation and coordination in software processes as its process modeling language enables the decomposition of complex processes into subprocesses that can be distributed and executed at different organizational sites. In GENESIS, workflow management technologies have been integrated with artifact management and communication services to meet the necessary requirements of managing the cooperation among distributed teams. Its strengths are a powerful activity management, covering all the main aspects of the life cycle of an activity; an efficient and flexible project monitoring, collecting productivity and quality metrics to show on-demand snapshots of the whole process and of its parts at different levels of detail, and a careful consideration of the process evolution questions, allowing to adequately manage the most common exceptions happening during process execution in a simple and flexible way. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: workflow management; process management; process modeling; software process improvement; artifact management
1. INTRODUCTION

The success of large software development and maintenance projects heavily depends on the attention project managers pay to the coordination, communication, and collaboration of the human resources involved in the project (Bandinelli et al. 1996). This requirement is becoming extremely important because more and more software organizations are moving from a traditional centralized software factory model to distributed virtual organization models, thanks to the new opportunities offered by Internet technologies (Maurer et al. 2000).

CASE tools typically provide a sequence of phases to be followed and some mechanism for refining a phase into more detailed phases. However, CASE tools only provide a ‘static view’ of the software development process that does not consider the practice of the developers, and consequently are inappropriate for the management and execution of the process (Chan and Leung 1997). Workflow Management Systems (WiMSs) (Georgakopoulos et al. 1995, Workflow Management Coalition 1994) and Process-centered Software Engineering Environments (PSEEs) (Ambriola et al. 1998, Grefen et al. 1999) are effective technologies for improving the management of software processes, particularly by improving the integration, coordination, and communication in a cooperative networking environment. The main difference between these two types of environments is the fact that WiMSs have been introduced for the modeling and automation of business and industrial processes, while PSEEs have been introduced for the management of software processes. Advantages of WiMSs are a reduced complexity and intrusiveness and a wider domain of applicability (Fuggetta 2000). However, despite the differences, PSEEs and WiMSs have many commonalities and in this article we will not distinguish between the two classes of systems; other authors refer to both with the term Process Support Systems (PSSs) (Casati and Cugola 2001).

Despite the advances made in the field, at the present workflow management technologies have mainly been applied to model, improve, and automate business and industrial engineering processes (Aversano et al. 2002a, Huang et al. 2000), while they have only marginally been adopted in the software industry (Aversano et al. 2002b). This has been mainly imputed to different facts:

- software processes are human intensive and are subject to deviations more than the more stable traditional engineering and administrative processes (Casati and Cugola 2001, Cugola et al. 1996);
- software engineers and project managers had the perception that the adoption of workflow management technologies in the centralized software factory model adopted in the past could only produce overheads in the management of the process without providing real advantages;
- PSEEs and WiMSs generally provide limited support to the management of the artifacts produced within a software process. Although some PSEEs explicitly consider software artifacts as process elements (Cugola 1998), they are scarcely integrated with configuration management and versioning tools that are considered of higher importance by industrial software engineers and project managers.

However, despite the skepticism of the past, the need of automatic support to the management of software processes is being recognized also in the software industry (Aversano et al. 2002b), because in modern virtual organization models people make substantially more use of network based communication than physical presence to interact and cooperate within a project. Indeed, modern WiMSs are web-based (Ames et al. 1997, Maurer et al. 2000) and provide facilities to coordinate distributed teams. However, although using the web as an infrastructure makes WiMSs more appealing in industry, the scalability and the real use of these systems depend on the ability to exploit distributed object technologies and middleware to enable the interoperability of workflow engines distributed across multiple organizational sites that operate both independently and as part of a single WiMS (Cugola et al. 2001).

In this article we present the GENESIS platform (GEneralised eNvironment for procEsS management in cooperative Software engineering), the outcome of a research project aiming at designing and developing a noninvasive and open-source system to support software engineering processes in a highly distributed environment. The system supports the cooperation and coordination in software processes as its process modeling language enables the decomposition of complex processes.
Research Section

GENESIS Environment

into subprocesses that can be distributed and executed at different organizational sites. In GENESIS, workflow management technologies have been integrated with artifact management and communication services to meet the necessary requirements of managing the cooperation among distributed teams. Its strengths are a powerful activity management, covering all the main aspects of the life cycle of an activity, an efficient and flexible project monitoring, collecting productivity and quality metrics to show on demand snapshots of the whole process and of its parts at different detail levels; and a careful consideration of the process evolution questions, allowing to adequately manage the most common exceptions and deviations happening during process execution in a simple and flexible way.

The article is organized as follows. Section 2 gives an overview of the GENESIS distributed platform and its subsystems. A description of the main features of the process modeling language is presented in Section 3, while project management is illustrated in Section 4. Section 5 discusses the experimentation and evaluation of GENESIS in research and industrial environments, while Section 6 compares GENESIS to related works. Finally, Section 7 concludes the article.

2. GENESIS OVERVIEW

GENESIS is a distributed environment supporting cooperative software engineering (Ritrovato and Gaeta 2002). Cooperation and collaboration in software engineering comprises all software engineering methods and tools that support teamwork flexibly and effectively. An environment that supports cooperative software engineering must fulfill at least the following features (Ben-Shaul and Kaiser 1995):

(a) policy-driven cooperation to manage the access to artifacts (policy of locking, sharing of source code and documents, etc.)
(b) informal cooperation to manage both structured and unstructured information (annotation for every artifact, automatic notification, etc.).

Both these issues are addressed in the GENESIS environment. In addition, GENESIS covers the communication and coordination requirements within a software process that are necessary for the planning, execution and coordination of all tasks-related, spatially and temporally distributed activities. GENESIS enables virtual organizations models by allowing cooperation, distribution and control of a software engineering project performed at different sites. The scenario is that of a project coordinator site taking the responsibility for the whole project, and a number of other sites managing, or contributing to, specific project work-packages. In general, one or more processes may be defined for each project or work-package at a site, and distribution of activities of these processes among other sites is possible. The GENESIS platform has been implemented according to the following guidelines that also match with the indications presented by Fuggetta (Fuggetta 2000) for future process-modeling languages:

- full autonomy of sites, i.e. freedom is given to each site to choose tools and software process models for the project;
- powerful Process Definition Language (PDL) that allows for a hierarchical decomposition of processes in terms of subprocesses, and their execution at different sites;
- intersite coordination based on distributed event engines.

A multilevels and distributed process consists of (atomic) tasks assigned to different sites where they are further modeled as subprocesses and possibly distributed in turn, as it is shown in Figure 1. Each intermediate level will be the coordinator process of all the tasks modeled and executed at a lower level. For simplicity, two-level processes will be considered in the sequel, so the terms 'global' and 'local' may be used without ambiguity. Subprocesses to be executed at a site are autonomously defined and modeled, the only requirement being that the interfaces specified in the global process model are respected. Each site locally executes instances of its own process models and interacts with the other sites mainly to exchange input and output artifacts.

All the GENESIS sites have the same architecture, depicted in Figure 2, which consists of different subsystems integrated through a common web client and coordination application logic layer. Concerning the technology used for the implementation, each GENESIS site is realized as a web application: the user interface and the coordination layer are realized using JSP and servlets (Tomcat being the...
The communication between the coordination layer and the different subsystems comprising a GENESIS site is based on Java RMI. The WfMS supporting database is based on MySQL Server. The transmission of events among sites is realized using HTTP and SOAP protocols to avoid security concerns with firewall settings of the different sites involved in a project. Figure 3 shows a snapshot of the GENESIS user interface.

The different subsystems comprising the GENESIS platform are described in the following sections.

2.1. Resource Management System

The resource management system manages the organization resources and their allocation to the projects. The basic functionalities of this module include:

- **User authentication to the GENESIS platform.**

  In the distributed setting, a user logs in the
GENESIS system of his organization only. This automatically opens his way through all the GENESIS sites he needs to communicate with for activities in distributed projects.

- **Management of the user data.** The information that the system maintains for each user consists of user’s personal data, his/her skills and his/her position in the organization.
- **Management of the organization projects.** The resource management system holds the projects list that the organization takes part in, and, for each distributed project, the list of the cooperating organizations.
- **Creation and management of the business position in the organization.** The resource manager can define the business positions of the organization. The access rights are associated with each position. A resource can be associated with one or more positions.

### 2.2. Artifact Management System

The artifact management system is responsible for creation, modification, deletion, and storage of artifacts. It provides facilities for indexing and searching the artifacts and collecting appropriate metrics (Boldyreff *et al.* 2002). Every artifact possesses a collection of associated meta-data, both standard and domain-specific, presented as an XML document. Conformance to a particular DTD indicates that the artifact is of a specific type. Several built-in types and some suitable defaults to store any type of artifact meta-data are provided by the system and creation of new artifact types is allowed through DTD extensions. Artifact types are also important for workflow management, being one of the elements of the GENESIS process-modeling language (see Section 3).

Artifacts are made aware of changes to related artifacts using notifications, allowing them to modify their own meta-data actively in contrast to other software repositories where users must perform all and any modifications, however trivial.

Support to collaboratively work on the same artifact is achieved through some access control policy. Two different access policies are supported: locking/unlocking and branching/merging. The former allows the worker to lock an artifact, so...
that he/she alone is allowed to create a new version of it. If someone tries to lock an already locked artifact, the system will notify him/her with the name of the user who has locked the artifact. The second policy allows the workers to create a new, branched version. Nested branches are allowed.

Drafts storage is also supported by the system. The draft is automatically locked and might undergo to several updates until the final version is released. Complete versions could be automatically reopened for updates if this is specified in the process model.

2.3. Workflow Management System

The workflow management system is in charge of the definition, instantiation, and execution of software processes. It consists of four main components sharing a common supporting database (see Figure 4). Specifically, the process definition tool and the project management tool allow to define processes and to monitor their execution, whereas the workflow engine and the work-list handler are the process enactment components.

The process definition in GENESIS happens at two stages. First, an abstract process model is created using the process definition tool. This model consists of a description of the workflow (including activities, actors, artifact types and transitions) independently of the particular project. Interaction with the artifact management system is required to define the types of artifacts to be produced by the activities. Then, the abstract process model is refined and customized in the context of a specific project through the project management tool, thus achieving a concrete (or refined) process model, which may be interpreted and enacted by the workflow engine. In fact, the concrete model also contains information such as the actual workers, the artifact names, scheduling etc. The project management tool interacts, through the coordination layer, with the resource management system to identify the human resources and the sites participating in the project for activities assignment. Moreover, this tool offers a number of facilities to modify the model if needed and for handling deviations during process enactment. For a distributed project, the project management tool implements a protocol for assignment of tasks to other GENESIS sites using the event engine, as described in subsection 4.2.

The distinction between abstract and concrete process models allows for reuse of the same abstract model in different projects. Furthermore, several process instances may be enacted from the same concrete model and they may evolve at runtime, independently of one another and of the originating model, as it will be explained in subsection 4.4.

A process visualization tool has been included to be used by the project manager both in the design phase, to get a picture of the process being modeled,
and during the enactment phase, to monitor process instances. The workflow engine reads the transition rules from the process definition and executes them, thus allowing the advancement of the process instance. In particular, the engine is in charge of creating new activity instances and assigning them to the workers specified in the concrete process model. The workflow engine interacts with the event engine to propagate events that are relevant for the execution of cooperating processes executed at other sites.

The work-list handler interacts with the database to retrieve information about activities and artifacts in the workflow for users. In particular, this component provides the worker with the list of activities assigned to him/her and is in charge of managing the life cycle of each activity. Some events, such as the activity termination or the production of an artifact during the activity, are forwarded to the workflow engine as they may trigger transitions in the process instance.

2.4. Event Engine and Communication Management System

The event engine is in charge of collecting events that arise within a process, in order to notify software components and users of the same site or other cooperating sites through the communication management system. Specific events have been used to allow synchronization and coordination of the process instances executed at different sites, and for dynamic reconfiguration of the whole software process.

GENESIS supports collaboration, allowing both formal and informal communication. Every worker can communicate with other workers, either of the same site or of different sites. To this aim, mechanisms have been integrated to support asynchronous and synchronous communication. GENESIS users can informally communicate through a chat tool (synchronous communication) or formally through an internal mailing system that allows sending/receiving messages with the possibility to attach files. Formal communication is also implemented by associating comments and annotations to an artifact, an activity, or a project and by using the GENESIS forum to publish and consult information. Messages and information concerning a project can be searched and retrieved by using different process elements, such as activities, artifacts, and people, thus maintaining a history of communications.

The internal mailing system also manages the message notification to the human resources whenever an important event occurs (i.e. an artifact is ready, a deadline passed, etc.). Some events are directly notified to interested people (e.g. a notification that an activity may be started is sent to the worker in charge of it); in general, notifications are made according to an event subscription policy.

2.5. Metric Engine

The GENESIS platform has been provided with a tool for monitoring the project enactment and collecting process metrics (Ballarini et al. 2003). GEM (GENESIS Measurements) is a software module in charge of periodically (batch mode) collecting metrics during the process and of presenting synthetic reports about the project status (on demand mode), allowing project managers to generate alarm events either manually or automatically.

The basic functionalities of GEM are to automatically collect data about the project status, to generate snapshots of the project, activities and artifacts, and to elaborate data for estimating completion rates. Both snapshots and measures are permanently stored, so that the project manager may have a view of the project evolution.

3. PROCESS MODELING IN GENESIS

In this section, we describe the process definition language (PDL) of GENESIS. The main elements of a process definition are: activities, actors, artifact types and transitions, which are described in the following sections.

The theoretical model for the transitions specification in GENESIS is the Event-Condition-Action (ECA) paradigm. The meaning is that during enactment, as soon as some predefined event occurs within the ‘source activity’, the (guard) condition is evaluated and some action is executed by the workflow engine on the ‘target activity’. Transitions in our model are used for both control flow and data (i.e. artifact) flow. This event-driven approach for the workflow management has turned out to be particularly suitable in the distributed setting, as these events may be easily propagated among, or
dispatched to, different workflow engines through the GENESIS event engine.

### 3.1. Modeling Activities

Activities are the work units of the process model. They may be interactive, i.e. requiring human intervention, automatic, i.e. automatically performed by a tool, or superactivities. Superactivities allow for a hierarchical decomposition of the process model as they correspond to subprocesses that will be modeled and performed at (possibly) different GENESIS sites.

In the abstract process model, activities have a name, the performing role(s) in case of an interactive activity or the tool-type for an automatic activity, and the output artifact types (artifacts to be produced and created ex novo). In fact, each activity of the process model is essentially described by the artifacts that it will produce, without additional information on how the activity should actually be carried out. Different artifacts of the same type may be produced within an activity, and there is no need for the process modeler to specify the multiplicity of these artifacts beforehand, as their number may not be known but may be dynamically defined during the process. On the other hand, the unitary multiplicity (i.e. exactly one) of an artifact can be specified by replacing the artifact type with the actual artifact name.

One or more variables may be attached to an activity or to an artifact type, whose values, generally assigned by the worker of the activity, will be an additional data of the process. Variables may be used to build conditions for routing the control or data flow of the process instance (see subsection 3.2).

### 3.2. Modeling Transitions

Following the standard formalism for ECA process-modeling languages, a transition is defined as a triple \((S, t, S')\), where \(S\) is the source of the transition, which in our case can be either an activity \(A\) or an artifact of type \(\alpha\) produced by \(A\); \(S'\) is the target of the transition, which will always be an activity \(A'\), and \(t\) is the transition label composed of a set of attributes in the form of \(e/c/a\), where \(e\) is the triggering event of the transition (occurred within \(A\)), \(c\) is the guard condition (the default value is ‘true’), and \(a\) is the action on \(A'\).

The following notation may be used for the transitions:

\[
I : S \xrightarrow{e/c/a} S'
\]

We have established a vocabulary for the event and action types currently supported by the GENESIS WIMS, and formulated a set of rules for the definition of the transitions. These rules are in part implemented by the process definition tool and the project management tool, which guide for a correct definition and consistency with respect to the basic constraints. Details of the language can be found in the appendix.

The event types we have considered so far are those essential for the advancement of the process execution; namely, the start of a (sub)process, closure or (abnormal) termination of an activity, and production or completion of artifacts. It is worth noting that the data flow here is always defined on the artifact types as, unless otherwise specified, several artifacts of the same type may be produced if needed. Also, different draft versions of the same artifact can be produced until the artifact is produced in a final version (completed). Intermediate versions of artifacts are considered for transitions where these drafts are read-only for the target activities.

Conditions are Boolean expressions on the workflow control data that permit or inhibit the transition. They involve variables associated with an activity or, in the case of a produced artifact event, with the artifact type source of that event. A condition is defined as a boolean expression composed of elementary relational expressions linked by the AND and OR operators. Each elementary relational expression may be a comparison of two variables or of a variable with a constant value.

The action attribute of the transition describes the system reaction to the occurred event in case of a true evaluation of the condition. Depending on the event raised, the action essentially consists of starting, or state changing for, an activity, or transfer of the source artifact (if any) to the target activity (ies) as read-only or for updates.

### 3.3. PDL Constructs

We have referred to the standard Unified Modeling Language (UML) notation of the activity diagrams for a graphical view of the transitions. This has been
enriched with symbols and data to represent the process model according to our (textual) PDL. An explanation of the definition and semantics of the main constructs with our PDL follows, which are graphically represented in Figure 5. For simplicity, some details related to transitions on artifacts have been omitted in the figure.

Two basic types of actions have been considered in the current implementation. We use the term 'flow' to express both actions, although their semantics depends on the transition event and on some other attributes as explained below.

- **Control flow sequence** (Figure 5(a)): this construct defines the execution order of two activities; the event is concerned with the termination of the first activity (A in figure); as an effect, the target activity (B) is started.
- **Data (and possibly control) flow sequence** (Figure 5(b)): An artifact of type $ar$, produced by activity A, is passed to activity B. Essentially, two action types may be used to model this construct, namely, dataflow only or dataflow with activation. If B is not yet started at the time the event is raised, with the first option an instance of B is created but this will be waiting for some other event before it can actually start; with the second option, instead, activity B is started by the same event.

In general, for transitions of data flow type the process modeler needs to specify whether the artifact to be produced is intended as input (i.e. read-only) or as input/output (i.e. update) for the target activity. In the latter case, the produced and closed artifact will be updated by the target activity.

Transitions caused by feedback artifacts are particular in that, as a consequence, the impacted artifacts might be reopened for corrections by the target activity. We have thought of a specific action type to this aim:

- **Feedback data flow** (Figure 5(i)): the triggering event is the production of a feedback artifact (of type $ar2$ by activity B in the figure). The action attribute must contain, in this case, also the artifact type to which the feedback refers ($ar1$). The effect will be that the feedback is received by A and the impacted artifact will be set to the openable state, i.e. a temporary state where the user may decide to reopen the artifact for updates.

Additionally, we have defined the following two action types that allow for a dynamic evolution of the process instance:

- **Fork and Nest** (Figure 5(c) and 5(c)'): the event is concerned with the production of a closed artifact of the specified type within the source

![Figure 5. Graphical view of the constructs](image-url)
activity (A in the figure); for the target activity (B in the figure), a new instance will be created by the workflow engine. So, subsequent productions of artifacts of that type by the source activity would lead to the creation of activity instances of the target activity, each consuming a different artifact. These instances may be running in parallel. In the figure, we have used the UML notation of * applied to the target activity to represent the Fork and Nest action.

- **Join and Reduce** (Figures 5(c) and 5(c)'); the event is concerned with the termination of the source activity (control flow join and reduce, Figure 5(c)) or the production of an artifact of the specified type by that activity (data flow join and reduce, Figure 5(c)'). In both figures, B is the source activity. This action is the dual of the Fork and Nest action. Its semantics is that all the activity instances of type of the source activity are joined into one instance of the target activity (C in the figure). For simplicity, in the figure the Join and Reduce arc has been used immediately after the Fork and Nest action, but this is not a constraint of the enactment layer.

The Fork and Nest operator is particularly useful when the number of artifacts of a given type produced by an activity cannot be planned at modeling time. An example is the ordinary maintenance process, analyzed in (Aversano et al. 2002b). This process has the production of a maintenance request (named trouble ticket) as the starting activity: for each request, the actual maintenance process starts, possibly on the client site. The artifacts ‘produced’ by the trouble ticket production activity are the clients’ requests forms, whose number cannot be specified beforehand. So, if A is the trouble ticket activity, the target activity B is modeled as a superactivity, associated with the same site or a different one. The corresponding subprocess is then modeled independently and will consist of some activities, such as request analysis, maintenance planning, anomaly resolution, etc. After each maintenance subprocess has ended, the ‘global’ workflow proceeds with the requests closure phase, where data on the resolved anomaly for each project is stored, and the maintenance requests will change state from open to closed.

Figure 5(d–h) shows some of the composite constructs. In particular, Figures 5(d) and 5(e) show the OR/AND split construct, i.e. 5d reads: after a closure/termination event by activity A, activity B1 is activated under condition c1 and activity B2 is activated under condition c2. Conditions c1 and c2 may be mutually exclusive; instead, 5e reads: each artifact of type ar produced/updated by activity A is passed to activity B1 under condition c1, and to activity B2 under condition c2.

Special attention should be paid to the synchronization constructs (AND join), shown in Figure 5(f–h). These are used to express the synchronization of events for the control or data flow (i.e. rendezvous of the produced artifacts at the target activity). In general, a synchronization condition may be attached to the bar, which will be evaluated every time one of the incoming transitions has fired. This condition is expressed as a boolean expression involving variables attached to the (concurrent) activities and/or artifacts, so that the control or data flow is blocked at the bar until the condition becomes true. This construct may also be used for a single transition, for example, to express a time constraint.

We have identified different types of synchronization, resulting from considerations on two facts: activity iterations and subsequent productions of artifacts of the same type by the same activity. For these reasons, in fact, concurrency of a group of activities may happen more than once during the process, and so the same synchronization construct may be applied several times after the occurrence of the same event types. A list of the occurred events from each activity is maintained for this aim by the workflow engine until the synchronization condition is satisfied at each iteration.

In this respect, the modeler is asked to specify whether the events synchronization should hold

- always: i.e. at each iteration, and so a precondition for a correct execution in this case is that the number of iterations of the concurrent activities is the same; or
- once: i.e. at the first iteration only, and then is relaxed.

In case of artifacts synchronization (Figure 5(g)), all the produced and closed artifacts for each type would normally be transferred to the target activity if the synchronization condition holds. Let us consider the example in Figure 6 (taken from (Aversano et al. 2003)), where the Coding activity produces several artifacts of type ar1, on which test
reports will be made by the Unit Testing activity (artifacts of type ar2).

Each time, the Consolidation activity should receive together the code artifacts and the test report referring to those artifacts. So, each group of code artifacts will be waiting for their successful test report to be transferred to the Consolidation activity. To model this fact, we have thought of a type of synchronization based on artifacts traceability. If this is the case, the type of the artifact and the producer activity with respect to which the dependency is originated, should be specified in the synchronization construct (the ar1 type and Coding activity in the example).

In general, the link among the artifacts to be synchronized may be expressed through variables attached to them. In this case, the synchronization condition would be that the values of these variables must be the same so that only the artifacts satisfying this rule, among those already produced, will be transferred together.

Collaborative work on the same artifact by different activities may also be specified in the model (Figure 5(j)). The artifact of type ar is sent to activities B1 and B2, which will independently produce branched versions of it. A subsequent activity of the model may eventually be required to merge the pieces (activity C in the figure).

3.4. Distributed Models

The abstract process model may contain superactivities. At this stage when the process design may be carried out without reference to a particular project, superactivities are simply interpreted as activities whose work is still complex and will need to be further modeled. This is generally used in a distributed context, where the executing organization may prefer to hide the internal structure of the superactivity to the partner organizations of the project. In fact, the conformance with the output artifacts specified in the global process will be the only constraint of the corresponding subprocess model.

The software process componentization through superactivities may be used to incrementally design the process in a top-down manner if, for example, its structure is not completely decided beforehand, and also enables integration of (sub) processes in a bottom-up fashion, thus leading to process models reuse.

The input transitions for the superactivity in the global model are replicated in the subprocess as shown in Figure 7 (where B is a superactivity and (b) is the model of the corresponding subprocess). However, any outgoing control flow transition from the superactivity will be determined by the end of the subprocess, and each outgoing data flow transition will be determined by the production of the artifacts by some activities of the subprocess (activity B3 will produce artifacts of type ar2 in the figure).

During the enactment, synchronization is needed for consistently maintaining the states of the superactivity at global level and of the corresponding subprocess at the local GENESIS site. In particular, when a superactivity is triggered by a control flow transition, this event is propagated through the event engine to the corresponding subprocess to trigger one or more local activities. On the other hand, the end of the local process produces an event.
that is forwarded to the global level to terminate the corresponding superactivity.

If an artifact that is input to a superactivity is produced, then this event is propagated to the corresponding subprocess so that the artifact can be transferred as input to some local activities. Similarly, the production of the local artifacts that are of interest at global level generates an event that is propagated to the global process.

4. PROJECT MANAGEMENT

When defining a process model for a project through the project management tool, an abstract process model is selected from those already available and it is enriched with the project-specific details to make it automatically executable by the workflow engine. However, the two models are decoupled so that changes may be made on the concrete model without affecting the original abstract model.

In the concrete model, roles are replaced by the actual human resources, tool types are replaced by the tools, and a naming convention is chosen for the artifacts. In case of an undefined multiplicity of these, the names of all the artifacts produced of some type will be decided by the author(s) completing a prefix assigned at this stage by the project manager.

For a multisite project, collaboration may be achieved through superactivities assigned to other sites, that will also keep the control over them, and through global activities, whose workers are from different sites. The project distribution follows a protocol described in Section 4.2.

The model refinement process may be incremental as the process can start as soon as the needed resources have been assigned to the initial activity, independently of the subsequent activities. In fact, checks are made at enactment time to make sure that the activities have their resources allocated when they need to be started.

4.1. Activity Management

Flexibility is guaranteed by several services provided to the user involved in the process execution; in particular, teamwork management services are provided at different levels. In this respect, interactive activities are always associated with groups of workers and may be labeled as single-user or collaborative. These teams are established on the basis of the particular skills required for the role, and they may overlap. Also, each team has a leader. A single-user activity will be normally performed by only one member of the group who volunteers for that (pool approach). However, the team leader will keep the visibility of the activity throughout its life cycle, he may directly assign the activity to a specific team member (push approach) or replace the assignee at any time.

Besides the list of output artifacts, a worker of an activity will be presented with other two lists of artifacts produced by the preceding activities, namely, input (or read-only) artifacts, and input/output (or updatable) artifacts. In the latter case, new versions of the artifacts will be released within the activity. As for output artifacts, different versions of an input/output artifact can be produced within an activity until the final version is produced.

For a collaborative activity, all group members may be required to work on the same artifacts. Synchronization facilities, such as locking/unlocking or branching/merging of artifacts, provided by the artifact management system are used in this case. Collaborative activities essentially provide a lightweight alternative to composite constructs of many parallel single-user activities producing branches of some artifact that will be merged in a subsequent unifying activity (Figure 5(j)). As this interactivities collaboration may be complex to handle at workflow level, and so is more sensitive to deviations from the model, we think that a coarse-grained definition through a single collaborative activity would probably be a better choice in some situation.

All activities have a state during enactment (see Figure 8). This is used by the system to monitor and manage the advancement of the process instance. The activity state change may be caused by actions occurring in the running process, which generally correspond to method invocations of the work-list handler component.

The scheduled state is assumed by all the activity instances as soon as the process starts. Interactive activities in this state are visible by the workers but cannot be executed until the necessary information is available.

The transition to the wait state is caused by the workflow engine to mean that the data is available for the activity, but this will be started after some other event. An example is that the incoming artifacts for the activity are collected as soon as
they are produced by other activities and before the activity is actually started through a control flow transition. The wait state is transparent to the GENESIS users.

In the activated state, the activity is ready to be started, thus the activity reaches the running state. A single-user activity is canceled from the work lists of the other group members except their leader. In case of a superactivity, the running state follows the start of the corresponding subprocess.

The worker of an interactive single-user activity may decide to suspend and later resume the activity, or simply abandon the activity (change to the released state) while this is in progress. Then the activity can be reassigned by the team leader and continued by someone else. The preempted state for the activity is reached when the team leader takes the activity to force its assignment to some worker. Normally, the activity will be closed by the worker after all the due artifacts have been produced or updated.

A collaborative activity may be started, suspended, resumed, and closed only by the team leader.

Superactivities are suspended, resumed, and closed according to the state of the corresponding subprocesses. These states are caused by actions of the project manager of the subprocess execution sites.

Any activity may be aborted in exceptional situations, thus reaching the terminated state. This state change may be a relevant event for the workflow progress and will be directly handled by the engine if the exception was foreseen in the model and a corresponding transition defined. Otherwise, a notification is sent to the project manager who may force a transition in the process model, thus causing a ‘deviation’ from the normal control flow of the process instance (see Section 4.4).

Reactivations (or iterations) of an activity usually happen in the case of cycles in the process model or, more generally, when the activity is closed, but later a new transition instance fires having that as the target activity, e.g. new input artifacts are sent by some previous activity of the model.

A log of all activities and process state changes exists and is used for metrics collection.

4.2. Creating Distributed Projects

The GENESIS platform has been designed to specifically support projects distributed among different sites. To this aim, the modeling and enactment of processes is also distributed. This has been possible owing to the PDL, which is general enough to respect the single organization rules, and thus allows keeping low the intrusiveness of the platform and is still powerful to assure coordination and control over complex software processes.

Three main phases have to be distinguished: the creation of the project identifying a coordinator site and the involved local sites, where the resource managers associate people to the project and select the project managers; the definition of the global process involving project managers of the different sites; and the definition of the local processes, independently defined by the different local project managers.

At the starting of a new project, the resource manager of the coordinator site creates a project using the resource management tool. This means that she/he selects the human resources allocated at the coordinator site and the local sites participating in the project. Then, each resource manager of an involved local site decides the allocation of the human resources and the local project manager.

Once the project has been created on the coordinator site, the project manager can start defining the required concrete process models for the project, starting from the available abstract process models (if a suitable abstract process model is not available, it has to be created first). Local project managers can collaborate with the project manager of the coordinator site for the definition of the global process.

Each superactivity has to be assigned by the project manager to a site participating in the project. In this case, information concerning the superactivity (start and end date, artifact types, etc.)
is sent to the local site and notified to the project manager.

The modeling is decentralized in that the process designers can edit subprocess models in an asynchronous way through the web-based user interface of GENESIS. The sites independently create the subprocess models, the only requirements being that the subprocess be available when the corresponding superactivity in the global process model has to be enacted, and that its interface (in terms of input/output artifacts) conform to that of the superactivity, as explained in Section 3.4.

For each global activity, the project manager of the coordinator site can select for each site the number of required people, and send this information together with the role associated with the global activity, and the work modality (collaborative/single user), to the project manager of the local sites.

A concrete global process can start independently of the local subprocess definition status at the participating sites. This allows for an incremental process definition and refinement at enactment time.

4.3. Project Monitoring

The platform support for project monitoring is provided by the process visualization tool and by the GEM tool. Both these modules interpret the process data recorded through the work-list handler component during execution.

In particular, the process visualization tool gives a snapshot of the process instance with the current data, highlighting the state of each activity, the artifacts produced, the current workers, etc. (see Figure 9). The GEM component considers 'on demand' the whole life cycle of each activity, identifying all the actions that have happened, the time and those responsible. The combination of this data with the estimation data on the duration of the process, effort for each activity, and artifact to be produced, previously stored through the project management tool, allows for an effective control of the scheduling constraints using an alert messaging mechanism.

4.4. Exception Handling and Deviations

Runtime facilities also include the management of the exceptions and mechanisms for the evolution of the process. Exceptions and failures are some of the basic issues addressed during the execution of a software process. In order to explain how exceptions are managed in GENESIS, a reference to the taxonomy proposed by Chiu et al. (Chiu et al. 1999) is made. The authors classify the exceptions
in two dimensions considering the source of the exception and the type of the exception. The source can be as follows:

- **External** when rising from external components participating in the WFMS such as the operating system, DBMS, software applications, machines and equipment, etc.
- **Internal** sources are related to workflow management issues such as the inability to find a resource or missing a deadline etc.

Exceptions can be of the following types:

- **Expected** if they are anticipated and already planned with explicit exception handlers.
- **Unexpected** if they require human intervention since it was unforeseen.

The exceptions we have considered in GENESIS are both the **internal expected** and the **internal unexpected** exceptions. As all the information about the current process instances are stored in the database, it is possible to manage the expected exceptions through ECA rules, where exception events may be defined and/or added to the PDL and the recovery actions specified in the transition description of the concrete model.

A simple mechanism to handle some **unexpected internal** exceptions consists of automatically sending notifications to the interested actors. This is applied, for example, when a transition fires but the target activity has no resource assigned yet, or, in case of a superactivity, the corresponding subprocess has not been modeled yet by the local project manager.

The unexpected exceptions can be normally handled by the project management tool owing to the possibility of managing deviations and evolving the process model at runtime.

In this respect, dynamic changes are allowed, which cover the most common corrective operations to a process, from simple adjustments to more complex changes to the activity net. In fact, the project manager can act on

- a single activity: changing state, resources, or the artifacts to be produced;
- a whole process: modifying the concrete model by adding or deleting activities and artifacts with the associated transitions, thus obtaining a new model in order for the running instances to better match the real process.

Also, the changes may be concerned with a single process instance or all of the running instances of that concrete model, e.g. modifying a team working on several instances of the same activity. In the first case, an ad hoc concrete model for that process instance would be created, while in the second case the changes are made directly on the old model. The project management tool implements mechanisms to ensure that the modified workflow meets the correctness criteria.

It is worth noting that the evolution of the processes can be effectively supported by the change operations policy, thus opening the way to incrementally build the process models. In fact, it is possible to first make a rough process model and execute its instances, then to detail the process model as soon as more knowledge and information are available, e.g. adding activities and transitions. The corresponding process instances will be automatically modified.

5. EXPERIENCE

The GENESIS project was carried out by three academic research institutions: CRMPA – University of Salerno (Italy), RCOST – University of Sannio (Italy), and the RISE research group of the University of Durham (UK), and three industrial organizations: MOMA srl (Italy), LogicDIS (Greece) and SchlumbergerSema (Spain). The three research institutions and MOMA developed the platform: in particular, RCOST developed the workflow management system, RISE the artifact management system, a research unit of MOMA and CRMPA located at the University of Rome (Italy), namely, GEM group, developed the metric engine, while CRMPA and MOMA developed the resource management system and the event and communication engine and also integrated the platform.

The two main industrial partners of the project, LogicDIS and SchlumbergerSema, acted as pilot users, providing real scenarios for the elicitation of requirements and validation of the platform. In fact, two simulated projects were run by the companies in the last four months of the project to experiment the platform. In order to avoid risks and facilitate the introduction of the platform in their environment, two projects that were already completed were executed again for these simulations, so to better analyze differences from using or not using the
platform in the same setting. It is worth noting that we were not able to integrate the artifact management system developed by RISE in the first version of the platform and, therefore, a simplified version developed by CRMPA was used in the experimentation. The simplified artifact management system provided basic facilities for storing and retrieving artifacts composed of one or more files, artifact versioning, locking/unlocking, and branching/merging.

In parallel to the pilot users validation, we decided to use the same version of the platform to host the last part of the GENESIS project, consisting of integration and testing of new functionalities and production and revision of the final deliverables. In fact, these activities were conducted by all the Italian research partners from their sites. The goal was to use the most advanced features of the platform, in particular, concerning the PDL constructs and distributed management, in order to better understand the platform limitations and lead to faster improvements and corrections on the way through.

Another opportunity to test the platform on a real project was offered by Elisys s.r.l, a small Italian company specialized in the production of electronic systems, including control and application software products. The GENESIS platform was chosen to conduct a case study for a methodology aiming to formally define all the processes of the company, and was used in a project to validate its formal definition against the actual process.

The GENESIS pilot users validation is summarized in Section 5.1; the GENESIS project internal experience is described in Section 5.2, and the platform use on real projects at Elisys s.r.l. in Section 5.3. In each context, a questionnaire was filled by all the participants, which provided us with a grounding for some qualitative data that we discuss in Section 5.4.

5.1. Pilot Users Validation

The definition of the GENESIS platform requirements, especially for the process-modeling facilities, followed a strict interaction with the pilot users of the GENESIS project. Both the industrial partners set up test-bed scenarios based on large-scale distributed software development processes they had used in real projects, to be able to validate all the essential requirements specified at the beginning of the project. The scenarios were enriched with tracked historic details such as incidents and troubles that occurred in the original projects, discussions about design and technical issues, coordination and communication mishaps, etc. Also, the same personnel as in the real projects participated in the simulations. The main reason for selecting these projects was that they provided sufficient complexity to justify distributed development teams and so were suitable for using and evaluating the platform.

The LogicDIS scenario reproduced the development of an Enterprise Resource Planning (ERP) system. The activities were carried out by three development teams located in three different sites. Two development teams were made of development units delivering cooperating modules/components of the ERP that would be later customized by the third team, according to the specific needs. The sites network topology consisted of a coordinator site based in Thessaloniky, which also implemented the modules customization subprocess, and two development sites based in Athens in different buildings. The GENESIS platform was evaluated against the actual procedure of the company.

The SchlumbergerSema scenario was based on a complex distributed software development project regarding a multimedia satellite-based communications system. In this context, the classical situation of shared and distributed software development (i.e. cooperative development of two software components of the project) was reproduced. The companies involved in the project were SchlumbergerSema and two subcontractors named ‘A’ and ‘B’ for sake of privacy. Company A was the coordinator site, in charge of the global management of the project and of the definition of the system requirements; SchlumbergerSema worked on software development of the Network Control Centre and also helped company A with the tuning of the system requirements; finally, company B developed the management of the database. According to the geographical distribution of the original project, a three-site architecture was used for the test bed, and some of the original personnel accepted to work on the simulation.
The two industrial partners evaluated the GENESIS platform from several aspects: workflow functionalities for the different phases of the project, artifact management, and communication. Some functionalities were judged to be more useful as they were nearer to their needs; in particular, the developers much appreciated the To-do List activity information section, the notification function, and the artifact-versioning system. Also, some non-functional aspects were considered, such as usability, reliability, performance, and user interfaces, although the tested version was still a prototype that was much improved for the final release.

The main advantages pointed out by the pilot users about the use of GENESIS were especially related to the distributed project management facilities, i.e. having a common view of the project and its status by all the involved partners, while keeping autonomy for local responsibilities. In particular, the SchlumbergerSema pilot users reported that some of the problems they had in the original project could have been avoided using the facilities of the GENESIS platform, in particular, concerning the automatic notification on events occurrence (major and minor project milestones); the internal communication mechanisms that could prevent misunderstandings between partners; the possibility to store messages in the artifact repository and attach them to deliverables; and the automatic update of the logbook.

5.2. Managing the GENESIS Project with GENESIS

An added value to the platform validation was also given by its internal usage for the design and execution of the remaining activities of the GENESIS project. In fact, this offered a real setting for an interorganizational process, with three different Italian partners, namely, CRMPA, RCOST, and GEM research groups, that carried out the last development activities of the platform for its final release and the project deliverables.

The distributed project consisted of three sites: CRMPA (located in Salerno) acted as coordinator, while RCOST (located in Benevento) and GEM (located in Rome) hosted the other two local sites. CRMPA was responsible for the integration of the new functionalities introduced in the different modules of the GENESIS platform, in particular, the workflow management system developed by RCOST, the metric engine developed by GEM group, and the event and communication system developed by CRMPA itself. The resource management system was not modified in the final phase of the development of the platform, while modifications to the artifact management system were not included in the experimentation of GENESIS. Moreover, the official project deliverables concerning the system architecture and the design and implementation of the platform were updated by all the partners in a collaborative way.

Several motivations lead us to this experimentation. Firstly, we wanted to test the platform in a mission-critical setting (the completion of the platform was crucial for the project success at the final review), while our industrial partners had preferred project simulations. Also, the time they spent on the training was saved in our context.

Secondly, our aim was to test the advanced features of the language, such as iterations, activities/processes with multiple instances, production of an unforeseen number of different artifacts of the same type, and the download/upload capabilities of the system for large artifacts (about 10 MB). In particular, some patterns for change management with feedback arrows and synchronization constructs were designed and directly tested. In fact, the testing activity of the integrated platform was collaboratively performed at the coordinator site, as a global activity, producing several change requests artifacts that were routed to the specific superactivities according to the impacted system components. This model caused a high degree of parallelism at both activity and subprocess level, where the tracking system for artifacts was highly desirable. This experience essentially lead to an extension of the language with specific parameters for re-opening artifacts, and also highlighted the need for more flexibility to allow recovery actions and dynamic changes on the process instance, features implemented for the final release.

Finally, we thought to evaluate by ourselves the benefits provided by an integrated environment for modeling, executing, and monitoring software development processes. Indeed, before using the GENESIS platform, the collaborative platform adopted for the GENESIS project was a web-based tool managing a centralized repository of artifacts, without configuration management and with a limited versioning support. The tool simply consisted...
of a publish–subscribe notification system integrated with the e-mail system that was useful in the first phases of the project to organize and review the project deliverables, owing to the possibility of attaching comments to the documents. However, this tool was not suitable to manage collaborative development of the system modules. As a result, each development team used his/her own versioning system for the code, with no global versioning and repository for the software artifacts, and independently chose the support tools, leading to a big effort spent for the integration of the first release. Because of the adoption of the GENESIS platform to manage the second release of the system, the effort made to design our process was compensated by the advantages of having an intersite activity coordination and collaboration; this guaranteed that everybody was working on the same versions of the code files and provided an increased awareness about who was modifying what and why.

5.3. Experience with a Real Project of a Small Enterprise

The introduction of the GENESIS platform at Elysis s.r.l. was for a different purpose, that is, supporting a lightweight approach for establishing a quality system at the company, so as to achieve the ISO 9001:2000 certification. This consisted of a method for redesigning processes and for introducing a software system to control and monitor the execution of processes.

The company had already developed an ad hoc software system to partially support the activities of the production processes related to the human resources management and the supply chain, but no process modeling and workflow automation facilities were provided. In general, the use of well-defined and documented processes could significantly improve the competitiveness of the enterprise. However, the definition and modeling of a process is a difficult task, if this has the form of a tacit knowledge as it is often the case for small to medium enterprises, and also resource consuming. The GENESIS platform was well accepted for this purpose, especially because of its operational flexibility and the noninvasive character. Also, the cost of acquisition of the platform was null, being an open-source software, while the training costs were low because some of the developers had acquired previous knowledge of the platform during a stage at the RCOST laboratories of the University of Sannio. A real project, aiming at the realization of a GSM system based on satellite technology was executed, which included the realization of software, hardware, and mechanical components. The automation and control of the documents’ flow through the platform allowed for the circulation in the process of stable documents only, and so reducing ambiguities that could arise from nonofficial or on-working deliverables.

For a Goal/Question/Metrics (GQM) (Solingen and Berghout 1999) based evaluation of the quality system methodology, questionnaires were distributed to the employees involved in the monitored project. Answers to specific questions concerning GENESIS aspects were also handled, and their analysis lead to results that contributed to the evaluation discussed in Section 5.4. It turned out that the answers concerning functionality and usability had been much influenced by the fact that an earlier version of the platform had been used, while people had greater expectations with respect to the support the tool was supposed to provide. More details can be found in (Aversano et al. 2004).

5.4. Evaluation Summary

The goals of the different experiences were different: the industrial pilot users of the GENESIS project experimented the platform in two pilot projects used as test beds to check its compliance to the requirements defined earlier in the project; the GENESIS development teams experimented the platform to test the most advanced features during the development of the second release of the platform itself, while the small Italian enterprise used the platform in a real industrial project to establish a quality system and to achieve the ISO 9001 certification. Only in the latter case was a GQM methodology followed that enabled to achieve quantitative results concerning the use of the platform (Aversano et al. 2004). From the other experiences, we only collected feedbacks from the users through a questionnaire that we can analyze in a qualitative way. Table 1 shows a representative subset of the questions of interest, with the answers provided by each pilot user. Each question refers to an attribute of four quality characteristics of interest, namely, functionality, reliability, usability, and efficiency. For each question, four options were available for the answers:
### Table 1. Questionnaire results

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Attributes</th>
<th>Questions</th>
<th>SS</th>
<th>LD</th>
<th>GE</th>
<th>EL</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1. Functionality</td>
<td>A1.1 Adequacy</td>
<td>Q1.1.1 The activity coordination facilities of the PDL are adequate to design your processes.</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q1.1.2 The project management tool covers the needs of your company.</td>
<td>B</td>
<td>B</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q1.1.3 Consider the To-do list: please indicate whether you think it is compliant with your expectations.</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q1.1.4 The distributed process management is adequate for your projects.</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>A1.2 Usefulness</td>
<td>Q1.2.1 With the GENESIS platform, we have experienced an increased software development productivity.</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q1.2.2 The GENESIS platform is helpful in supporting developers.</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q1.2.3 The feature that allows the visualization of the process graph is useful during process modeling.</td>
<td>–</td>
<td>A</td>
<td>A</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q1.2.4 What is your impression about the effectiveness of the instant messaging feature integrated in the platform.</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q1.2.5 In summary, the GENESIS workflow and resource management systems are useful for project managers and process designers.</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q1.2.6 In summary, the GENESIS workflow and resource management systems are useful for workers.</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>C2. Reliability</td>
<td>A2.1 Robustness</td>
<td>Q2.1.1 The interface interaction mechanism prevents errors.</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q2.1.2 It is easy to terminate or cancel any operation at any time.</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>A2.2 Error recovery</td>
<td>Q2.2.1 After an error, the interface allows for an easy recovery.</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q2.2.2 The GENESIS platform provides sufficient features for error recovery.</td>
<td>C</td>
<td>B</td>
<td>D</td>
<td>C</td>
</tr>
<tr>
<td>C3. Usability</td>
<td>A3.1 Learnability</td>
<td>Q3.1.1 It is easy to provide the system with input data.</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q3.1.2 It is easy to get output data from the system.</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q3.1.3 It is easy to search for data.</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q3.1.4 It is easy to learn the main procedures.</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>A3.2 User interfaces</td>
<td>Q3.2.1 Interface components are well organized on the screen.</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q3.2.2 Terms denoting commands are clear.</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q3.2.3 It is easy to learn and remember single interface component roles.</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>A3.3 Flexibility</td>
<td>Q3.3.1 The GENESIS platform does not require extensive modifications to our practices in order to be used.</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q3.3.2 How would you rate the usability of the process definition tool and the project management tool.</td>
<td>C</td>
<td>B</td>
<td>D</td>
<td>C</td>
</tr>
<tr>
<td>C4. Efficiency</td>
<td>A4.1 Response time</td>
<td>Q4.2.1 The performance of the GENESIS platform during a process execution is good.</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>A4.2 Performance</td>
<td>Q4.2.2 The performance of the GENESIS platform for the artifact management (e.g. upload/download of artifacts) is good.</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>–</td>
</tr>
</tbody>
</table>

SS, SchlumbergerSema; LD, LogicDIS; GE, GENESIS; EL, ELISYS.
A. ‘I totally agree’, or ‘Excellent’
B. ‘I agree’, or ‘Good’
C. ‘I disagree’, or ‘Fair’
D. ‘I totally disagree’, or ‘Unsatisfactory’

From the table, we can read quite encouraging results for functionality. In fact, the platform resulted in being fully compliant with the functional requirements of the pilot users; however, not all the potential of the platform was tested by the industrial partners (in particular, concerning the PDL). This was especially true for Elisys s.r.l., where an even earlier version with less functionalities was used, for example, without the visualization tool and the support for distributed execution of processes. Also, the visualization tool could not be used by SchlumbergerSema, and the reason for that was not really clear. More severe answers came from the GENESIS internal experimentation, as this helped us to provide insights into the specific features to be modified/added for the final release. The versioning support for both global and local artifacts revealed to be time saving.

In all cases, a progress with respect to the older procedures was observed, and the perception that the platform could be easily introduced in the organization and effectively used to support software development processes in all their facets was rather general.

Less positive results concern usability and reliability. From the comments of the pilot users, it is clear that although the user interface is easy to understand, it could be improved by providing more global views with all the process information at the same time. Also, the overall modeling process could benefit from a graphical editor as in modern workflow management systems. Little support for errors recovery was noticed, although this did not much affect the correct execution of the processes that all came to their end. An effort to improve reliability was also spent for the final release.

6. RELATED WORK

Research on workflow management and PSEEs is long dated. Different process-modeling languages have been defined in the workflow management literature. Some of them focus on the specification of the control flow between activities (Grefen et al. 1999, Workflow Management Coalition 1999) and express other enactment rules, in particular, concerned with exception handling, using the triggering facilities of the supporting database (Casati et al. 2000). Other approaches are interested in modeling the data flow of documents and artifacts produced within the process (Medina-Mora et al. 1992). GENESIS WfMS also adopts the latter approach, as this is more suitable for applying workflow management technologies in software engineering processes. PSEEs also explicitly consider artifact production as triggers for workflow enactment and model them within the process. For example, some of them use colored Petri nets as a basic formalism for process modeling and enactment (van der Aalst 1998, Bandinelli et al. 1996). Other approaches model a process as activity nets with expression of the data flow (Cugola et al. 1995, Heumann et al. 1996).

Most of the WfMSs and PSEEs of the last decade are client-server systems, with centralized enactment facilities, although they do not exploit the web as basic infrastructure to ease the accessibility by remote users. Recent research on workflow management is focusing on the use of web technologies. Like GENESIS, MILOS is a web-based WfMS supporting dynamic coordination of distributed software development teams over the Internet. MILOS allows for dynamic changes on the project plan during project enactment and provides a notification mechanism that tracks product changes during project enactment and informs the concerned people of the change. The workflow control was centralized in the first version and then a strategy for a distributed process enactment was proposed (Kötting and Maurer 1999), where different portions of a process model and data are stored on different workflow engines communicating through a peer-to-peer protocol. In addition to GENESIS, the mechanisms provided by MILOS are based on the specification of the interfaces between process and subprocesses. Recent work on MILOS is toward supporting distributed agile software projects (Bowen and Maurer 2002).

In the Ozweb environment, the peer-to-peer paradigm for distribution is adopted (Ben-Shaul and Kaiser 1995, Kaiser et al. 1998). In this environment, a decentralized system consists of independent subsystems spread among multiple sites. In particular, the authors focus on the process autonomy of each subsystem that should be self-contained and operationally independent. To this aim, they introduce the concept of ‘treats’ to
guarantee compliance of the artifacts exchanged between subprocesses. In GENESIS, the interface between subprocesses is also specified through the exchanged artifact types.

The Endevors system (Kammer et al. 1998) uses a layered object model to provide for the object-oriented definition and specification of process artifacts, activities, and resources. A coordination mechanism for distributed process execution and tool integration that uses the Hypertext Transfer Protocol (HTTP) is provided. The intent of distribution is to support a wide range of configurations with varying degrees and kinds of distribution. Stand-alone configuration with a base system without distributed components and multiuser configuration with a single remote data store are the configurations experimented for distribution. GENESIS also uses HTTP and SOAP for the communication of events among different sites during distributed process definition and enactment.

Eder and Panagos (Eder and Panagos 1999) propose an approach for the distributed execution that exploits the central role of an event notification service, READY. Workflow participants, both workflow engines and agents, can subscribe to events that trigger the start of workflow activities and processes, and events that describe state changes in the workflow processes they are interested in. Therefore, the configuration of the participants in a workflow can be dynamically changed without requiring any modifications to the existing architecture. Moreover, time-related constructs for addressing the time aspects of process management are provided. In GENESIS, a distributed event notification service is used.

PROSYT is an artifact-based PSEE (Cugola 1998). Each artifact produced during the process is an instance of some artifact type, which describes its internal structure and behavior. All the routing in this model is based on the artifact and the operations on them. Boolean expressions are used to express the constraints under which operations are allowed to start. PROSYT also allows for distributed enactment facilitated by the JEDI event-based middleware (Cugola et al. 2001); the same middleware is also used by the OPSS WfMS (Cugola et al. 2001). However, unlike MILOS and GENESIS, the lack of an activity-based view of the process does not allow for a hierarchical decomposition of a software project, which is generally required for project management purposes.

The systems discussed so far do not address the problem of decentralized process modeling. In GENESIS, facilities for collaborative editing of distributed software processes are provided. Grundy et al. (Grundy et al. 1998) also focus on problems concerning the distribution in process modeling. The proposed system provides mechanisms for collaboratively editing process models both in a synchronous and asynchronous way, together with version management support. The architecture is based on a central site maintaining the process model and distributed sites enacting portions of the model.

Another characteristic of GENESIS is that a process model can be defined at two abstraction levels. This means that there is an one-to-many correspondence between abstract models and concrete models, thus increasing the possibility of reusing process models. Also, different process instances can be executed with respect the concrete process model, as it is in most workflow management systems. Most of the software process support systems do not provide this separation between models at different abstraction levels, and, in some cases, there is a one-to-one correspondence between process models and process instances. In most cases (e.g. (Cugola et al. 2001, Heimann et al. 1996, Maurer et al. 2000)), process instances are populated copies of the models. This enables the runtime deviations of a single process instance without affecting the process model. Owing to the one-to-many correspondence between concrete process models and process instances, in GENESIS a copy of a concrete process model is only made in case of deviations of a process instance. This enables addressing ad hoc modifications of a process instance like in other approaches, as well as runtime modifications of a concrete process model, causing a selective modification of the related process instances. A similar approach is also used in WIDE (Grefen et al. 1999), while Heimann et al. (Heimann et al. 1996) use graph-rewriting rules to enact and change the process model at runtime.

Besides runtime modifications of a process model, GENESIS also provides mechanisms to increase flexibility during process enactment. Other authors also recognize the importance of providing flexible enactment mechanisms. In particular, PROSYT users are not forced to satisfy the constraints stated in the process model (Cugola 1998). They can invoke operations even if the associated constraints are not satisfied. PROSYT keeps track of the results of these
deviations and controls the invariants so that they are not violated as a result of such deviations. In GENESIS, flexibility is provided by distinguishing between the roles of the process actors. In particular, features are provided for a flexible management of activities, such as pull mechanisms that enable members of a group to accept activities, as well as features for team leaders to dynamically change the person in charge of an activity. Flexibility is also provided through the maintenance of a degree of indeterminism on the number of artifacts of a given type produced by an activity, thus giving the possibility of deciding this at runtime without the need of changing the process model. Also, it is possible to create at runtime an undefined number of instances of a given activity, depending on the artifacts produced during the process.

Finally, the integration with an artifact repository is another important aspect provided by GENESIS. Other systems, like PROSYT (Cugola 1998) and OZWEB (Ben-Shaul and Kaiser 1995, Kaiser et al. 1998) are artifact based and explicitly address the management of artifacts. However, it is not clear if configuration management facilities are provided or integrated in these systems. GENESIS integrates both an activity-based workflow management system and an artifact management system that provides versioning facilities and the cooperative and concurrent modification of software artifacts.

7. CONCLUSIONS

In this article, we have presented the GENESIS (GEneralised eNvironment for procEsS management in cooperatIve Software engineering) platform, the outcome of a European research project aiming at designing and developing a noninvasive and open-source system to support software engineering processes in a highly distributed environment. In fact, the GENESIS project itself was realized by three academic research institutions and three industrial partners, two of them playing the pilot users. The platform implementation was conducted by four development teams from different sites (each team mainly devoted to a different subsystem), which strictly cooperated and interacted through the coordinator site. The project started in September 2001 and ended in November 2003. The total effort was of 294 man-months and about 104,000 lines of code have been produced.

In this article, we focused on the workflow and project management components; in particular, we have described the facilities for process definition and flexible enactment of distributed processes. The process definition language is based on the event-condition-action paradigm that is common to many WfMS currently used in industry, but we have also added some new features. We have defined a two-layered view of the process, distinguishing an abstract model as a reference model for project managers and a concrete model that also contains details related to a specific software project and it is suitable to be enacted by the workflow engine. A multilevel process definition is obtained through superactivities, which correspond to subprocesses at a lower level and can be assigned and executed at different sites. This choice preserves the autonomy of each site during the process modeling and allows reusing the same PDL in all the GENESIS sites, thus simplifying the sites’ interoperability and communication within distributed projects. Also, the modeling process may be incremental, and refinements to the model may be made during enactment.

The definition of the GENESIS platform requirements for distribution, especially for the process-modeling facilities, followed a strict interaction with the pilot users of the GENESIS project. During the project, we collected descriptions of their work modalities on distributed processes and emerging needs. We considered the problems expressed by them and translated them into formal requirements for the implementation of the GENESIS platform.

Additionally, we have tried to fulfill the scientific objectives of the GENESIS project, such as noninvasiveness, i.e. the platform supports software engineering processes without interfering with the organization practices; dealing with cross-organizational multicompany projects, by providing facilities for collaboration, communication, coordination, and monitoring at global and local levels where each partner may keep its own development practices; and support for process evolution. These features have been evaluated through four different experiences: two of them were performed by the pilot users of the GENESIS project, which executed simulated projects; another experience consisted of using the platform to model and execute the remaining activities in the last four months of the GENESIS project itself; and the fourth experience was to use the platform in a real project of a small Italian
enterprise, with the aim of testing an approach for establishing a quality system at the company. Qualitative data from these experimentations have been discussed in the article and the overall judgment seems quite encouraging.

All the partners of the GENESIS project have shown their intent to further evaluate the platform through its usage in real industrial or research projects. In particular, some of the academic institutions are trying to introduce it in their teaching activities, for example, to control and manage student projects within the different software engineering courses. The GENESIS platform has been delivered as an open-source software and can be downloaded from the SourceForge website. In fact, we expect and would much appreciate contributions from other development communities. We also aim at continuing the evolution of GENESIS to further enhance some of the characteristics that achieved more attention by the evaluators.

APPENDIX

The formalism we present here for the transitions has been defined generally. However, the specification rules apply to the class of events and actions we have considered in the current implementation, which covers the essential requirements of a distributed process modeling. The generality of the formalism is well suited for extensions of the specifications to larger settings.

Setting

We briefly recall the notation already presented in Section 4.2, and explain some of the rules for the definition of the transitions.

We first need to introduce notations for the event, condition and action parts of a transition. Almost all sets considered are extended with 0, a special symbol used in relations to mean that no element of the corresponding set is specified.

\[ E \] is the set of event types, with \[ E \supseteq \{ \text{start subprocess}, \text{terminated (activity)}, \text{closed (activity)}, \text{produced (artifact)}, \text{completed (artifact)}, \text{external artifact} \} \]. The meaning of the specified events is contained in Table A1.

Let \( A \) be the set of actions, with

\[ A \supseteq \{ \text{Cfsequence}, \text{Dfsequence}, \text{UDFsequence}, \text{DFsequence} \& \text{act}, \text{UDFsequence} \& \text{act}, \text{ForkAndNest}, \text{CFJoinAndReduce}, \text{DFJoinAndReduce}, \text{UDFJoinAndReduce} \} \].

### Table A1. Events

<table>
<thead>
<tr>
<th>Event type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start subprocess</td>
<td>An event of this type may arise in the global process and be forwarded to a local site to start a subprocess.</td>
</tr>
<tr>
<td>Closed (activity)</td>
<td>An event of this type is raised after the user has terminated the activity.</td>
</tr>
<tr>
<td>Terminated (activity)</td>
<td>An event of this type is raised whenever the activity has been terminated abnormally (e.g. aborted).</td>
</tr>
<tr>
<td>Produced (artifact)</td>
<td>An event of this type is raised whenever a version of an artifact (including drafts) of the specified type is produced.</td>
</tr>
<tr>
<td>Completed (artifact)</td>
<td>An event of this type is raised whenever a version of an artifact that is not a draft, of the specified type, is produced.</td>
</tr>
<tr>
<td>External artifact</td>
<td>An event of this type is passed from the global process when an artifact of the specified type has been produced and is needed for the subprocess.</td>
</tr>
</tbody>
</table>

### Table A2. Actions

<table>
<thead>
<tr>
<th>Action type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cfsequence</td>
<td>Sequence of the control flow.</td>
</tr>
<tr>
<td>Dfsequence</td>
<td>Sequence of the data flow as read-only (DF) or update (UDF) with respect to the target activity.</td>
</tr>
<tr>
<td>UDFsequence</td>
<td>Sequence of the data flow (read-only or update) with activation of the target activity.</td>
</tr>
<tr>
<td>Dfsequence&amp;act</td>
<td>Sequence of the data flow as read-only (DF) or update (UDF) with activation of the target activity.</td>
</tr>
<tr>
<td>UDFsequence&amp;act</td>
<td>Sequence of the data flow as read-only (DF) or update (UDF) with activation of the target activity.</td>
</tr>
<tr>
<td>ForkAndNest</td>
<td>Dynamic fork of multiple instances from the same activity definition, each originated by an artifact production event by the source activity.</td>
</tr>
<tr>
<td>CFJoinAndReduce</td>
<td>Join of all the activity instances of the same activity definition, after closure of these instances.</td>
</tr>
<tr>
<td>DFJoinAndReduce</td>
<td>Flow of all the artifacts produced by activity instances of the same activity definition into one instance of the target activity. These artifacts may be read-only (DF) or update (UDF) for the target activity.</td>
</tr>
<tr>
<td>UDFJoinAndReduce</td>
<td>Join of all the activity instances of the same activity definition, after artifact production events by these instances, where the artifacts are read-only (DF) or update (UDF) for the target activity.</td>
</tr>
</tbody>
</table>

\( \text{CFJoinAndReduce}, \text{DFJoinAndReduce}, \text{UDFJoinAndReduce} \). (see Table A2).

Let \( V_{\text{art}} \) be the set of all artifact variables, \( V_{\text{art}} \) the set of all artifact variables, \( \text{Const} \) a set of constants, and \( \text{Op} \) a set of operators. Let \( V \subseteq V_{\text{art}} \cup V_{\text{art}} \).

Copyright © 2004 John Wiley & Sons, Ltd.
A condition \( c \) is defined as \( c = c (V, \text{Const}, \text{Op}) \) or \( c = \text{‘true’} \) if \( V = \{0\} \). Without going further into technical details, a condition is defined as a boolean expression composed of elementary relational expressions linked by the AND and OR operators. A relational expression may be a comparison of two variables or of a variable with a constant value. We shall denote the set of conditions by \( C \).

**Definition**

Let \( \text{Act} \) be the set of activities, \( \text{Art} \) the set of artifact types, and \( T \) a set of labels.

A transition is defined as a triple \((S, t, S') \subseteq (\text{Act} \times \text{Art}) \times T \times \text{Act}\), where \( S \) is the source of the transition, which can be either an activity \( A \) (e.g. \( S = (A, 0) \)) or an artifact of type \( \alpha \) produced by \( A \) (\( S = (A, \alpha) \)), or else an artifact produced outside the process (\( S = (0, \alpha) \)); \( S' \) is the target of the transition, which will always be an activity \( A' \), and \( t \) is the transition label composed of attributes in the form of \( e/c/a \), where \( e \in E \) is the triggering event of the transition (that occurred within \( A \)), \( c \in C \) is the guard condition (the default value is ‘true’), and \( a \in A \) is the action on the activity \( A' \).

The following notation may be used for the transitions:

\[ t : S \xrightarrow{e/c/a} S' \]

A set of rules exists for the definition of the transitions, to make sure, for example, that whenever an artifact is specified as source of the transition, the event and action parts refer to the data flow. Also, if \( e = \text{produced} (\text{artifact}) \), then \( a \notin \{ \text{UDFsequence}, \text{UDFsequence}\&\text{act}, \text{UDFJoinAndReduce}\&\text{act}, \text{UDFJoinAndReduce} \} \), i.e. a draft cannot be transferred to another activity in update mode. Other rules are concerned with composite constructs like activity or artifact synchronization.

All these rules are in part implemented by the process definition tool and the project management tool, which guide for a correct definition and consistency with respect to the basic constraints.

**REFERENCES**


